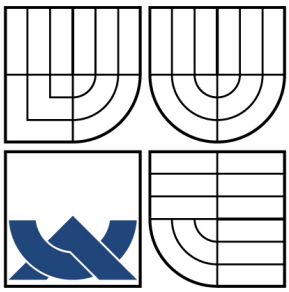
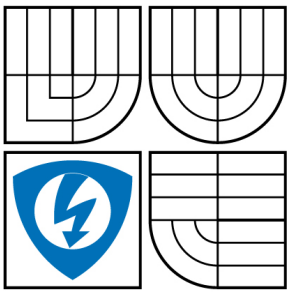


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

ŘÍZENÍ KROKOVÝCH MOTORŮ S CPLD

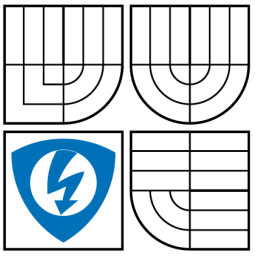
STEPPER MOTOR CONTROL USING CPLD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN DANĚK
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL KOVÁČ
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Bakalářská práce

bakalářský studijní obor
Elektronika a sdělovací technika

Student: Jan Daněk ***ID:*** 73068
Ročník: 3 ***Akademický rok:*** 2008/2009

NÁZEV TÉMATU:

Řízení krokových motorů s CPLD

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte principy řízení různých typů krokových motorů. Dále prostudujte funkci obvodů CPLD a způsob jejich programování ve VHDL. Navrhněte kontrolér v CPLD pro konkrétní typ krokového motorku.

Navrženou konstrukci odsimulujte a nakonfigurujte do obvodu CPLD. Funkční zapojení realizujte v podobě desky plošného spoje. Hotové zapojení doplňte dálkovým řízením motoru pomocí vhodně zvolených RF modulů.

DOPORUČENÁ LITERATURA:

[1] PERRY, L. D. VHDL: Programming by Example, 4/E. McGraw-Hill, 2002.

[2] ŘEZÁČ,K. Krokové motory [online]. 2002 - [cit. 7.5. 2008]. Dostupné na www: http://robotika.cz/articles/steppers/cs

Termín zadání: 9.2.2009 ***Termín odevzdání:*** 5.6.2009

Vedoucí práce: Ing. Michal Kováč

prof. Dr. Ing. Zbyněk Raida
Předseda oborové rady

UPOZORNĚNÍ:

ANOTACE

Tématem mé bakalářské práce je řízení krokových motorů s CPLD. Vytvoření radiově řízeného anténního rotátoru. Řízení je prováděno osobním počítačem pomocí RF obvodů. Práce obsahuje objasnění konstrukce a principu krokových motorů. Základní strukturu CPLD obvodů a její aplikaci pro vytvoření řídícího obvodu. Probrány jsou možnosti snímání polohy, uložení dat obvodem CPLD a radiového spojení. Výsledkem práce je návrh řídícího programu v jazyku VHDL, vytvoření zařízení realizovaného pomocí vývojové desky. K vývojové desce je připojen radiový přijímací obvod a budící obvod krokového motoru. Vysílací část je samostatně připojená k osobnímu počítači nebo jinému zařízení pomocí rozhranní RS232. V práci jsou navrženy desky plošných spojů vysílací a přijímací části. Pro osobní počítač je vytvořen program pro přenos dat komunikující s RS232.

ANNOTATION

The aim of my bachelor thesis was the control of CPLD stepper motor. To create a radio controlled antenna rotator. The controlling is done by a computer using RF circuits. The thesis includes a construction and principles of stepper motor. Basic structures of CPLD circuits were applying to create the control circuit. The possibilities of sensing the position and the data storage circuit were discussed. The control program in VHDL was implemented in development boards. The development board is connected to radio receiver circuit and the stepper motor driver. A transmitting board was connected to a personal computer or other device using RS232 interface. Printed circuit boards for transmitter and receiver circuits were created. The PC program was designed for data transmission. The program sends data and communicates via RS232 interface.

KLÍČOVÁ SLOVA

Řízení krokových motorů, CPLD, VHDL, stavový automat, RS232 ,RF

KEYWORDS

Stepper motor control, CPLD, VHDL, state machine, RS232, RF

DANĚK, J. *Řízení krokových motorů s CPLD.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav radioelektroniky, 2009. 25 s., 19 s. příloh. Bakalářská práce. Vedoucí práce: Ing. Michal Kováč.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Řízení krokových motorů s CPLD jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících, autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 19. května 2009 .....
podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Michalu Kováčovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne 19. května 2009 .....
podpis autora

OBSAH

SEZNAM OBRÁZKŮ:

SEZNAM TABULEK:

ÚVOD

Cílem této bakalářská práce je vytvoření anténního rotátoru řízeného bezdrátově z osobního počítače.

Pro pohon rotátoru byl vybrán unipolární krokový motor. Ten se oproti bipolárnímu vyznačuje malým odběrem a menším kroutícím momentem. Pro návrh a aplikaci řídícího obvodu krokového motoru byly vybrány programovatelné logické obvody (CPLD). Ty umožňují celé řízení vložit do jediného obvodu, který přijímá řídící signály. Na jejich základě provádí spínání cívek krokového motoru. Tento obvod je řízen vloženým programem. Program je vytvořen v jazyku VHDL ve vývojovém prostředí Xilinx ISE Design Suite 10.1. V aplikaci je použita vývojová deska XC2-XL firmy Xilinx. Naprogramovaný obvod CPLD je doplněn o budící obvody cívek a radiový přijímací obvod pracujícím na frekvenci 433 MHz .

Principy funkce a řízení krokových motorů jsou uvedeny v kapitole 1. Popis druhů a funkce obvodů CPLD obsahuje kapitola 2. Popis doplňujících obvodů je v kapitole 3. Konstrukční řešení zadání je v kapitole 4. Celkové hodnocení je umístěno v kapitole 5.

1. KROKOVÉ MOTORY

1.1 Funkce motoru

Motor je obecně zařízení přeměňující energii elektrickou na energii mechanickou. Elektrické motory pracují nejčastěji na principu silového působení elektromagnetických polí. Toto působení popisuje především Lorenzova síla, Ampérova síla [1].

Motor můžeme rozdělit na dvě hlavní části: rotor (pohybující se část) a stator (pevná část). V praxi je uspořádání takové, aby magnetické pole statoru a rotoru vytvářelo kroutící moment přenášený na rotor motoru. V současné době existuje několik základních principů elektromechanických motorů [2]:

Komutátorové elektromotory

Jeden z prvních motorů tohoto principu vynalezl Michael Faraday v roce 1821. Tyto motory jsou napájeny stejnosměrným proudem. Komutátor je součást zajišťující změnu směru procházejícího proudu a tím i změnu magnetického pole. Tato změna nastává dvakrát během jedné otáčky (po 180°).

Rychlost motoru závisí na velikosti napětí a proudu procházejícího cívkou motoru a na velikosti zátěže. Točivý moment je úměrný proudu. Regulací napětí regulujeme také otáčky. Výhodou stejnosměrných motorů je jednoduchá konstrukce.
Změnu směru se provádí přepólováním napájení. To neplatí pro sériový a derivační motor, kde je nutné přepólovat jen jednu z cívek.

Stejnosměrný motor s permanentním magnetem

Je dnes používán především pro malé výkony a aplikace na stejnosměrný proud.

Sériový elektromotor

U větších motorů je rotor vytvořen pomocí elektrického obvodu. Tento obvod je tvořen cívkou (vinutí rotoru) zapojenou do série s budící cívkou statoru (vinutí statoru). Hlavní výhodou sériového elektromotoru je nepřímá úměrnost točivého momentu na otáčkách. Při rozjezdu (nulové otáčky) má pak největší kroutící moment. Používají se pro dopravní stroje.

Derivační elektromotor

Konstrukce je obdobná sériovému motoru, ale cívka rotoru je zapojena paralelně se statorem. Otáčky motoru pak nejsou závislé na zátěži motoru. Používají se u strojů, kde jsou vyžadovány neměnné otáčky.

Sériový a derivační elektromotor pracují nejenom na stejnosměrný proud, ale i na proudy střídavé malých frekvencí. Mohou dosáhnout širokých rozmezí otáček na rozdíl od střídavých elektromotorů. Nevýhodou je samotný komutátor. Spíná velké proudy, je mechanicky namáhán, je zdrojem rušení.

Bezkomutátorové elektromotory

Jsou motory založené na vzniku točivého magnetického pole.

Synchronní motor

Rotor je tvořen permanentním magnetem. Stator tvoří cívky, jimiž protéká střídavý proud a vytváří rotující magnetické pole. Rotor se snaží udržet polohu tohoto magnetického

pole a rotuje. Motor udržuje pravidelné otáčky řízené frekvencí střídavého proudu. Nevýhodou je potřeba rozběhnutí motoru při zapnutí (rozběhové vinutí, jiný stroj). Při vysoké zátěži motor vypadne ze synchronizace a zastaví se.

Asynchronní motor

Rotor se skládá ze sady vodivých tyčí ve tvaru válcové klece. Tyče jsou na koncích vodivě spojeny (kotva nakrátko). Pro stojící motor indukuje rotující magnetické pole statoru v tyčích rotoru elektrické proudy. Ty vytváří vlastní magnetické pole. Magnetická pole rotoru a statoru navzájem reagují a vzniká magnetický moment. Otáčky rotoru vzrůstají. Když se přibližují otáčky rotoru otáčkám magnetického pole, klesají indukované proudy. Klesají také otáčky rotoru a tím i točivý moment motoru. Pokud je motor zatížen, nedosáhne otáček daných frekvencí napájecího proudu. Proto název asynchronní motor.

Speciální případy elektromotorů

Lineární elektromotor

Vícepólový elektromotor se statorem rozvinutým do přímky. Používá se pro velmi jemné posuvy.

Střídavé servomotory

Jsou motory vytvořené pro pohony. V principu jde o synchronní motory s permanentními magnety v rotoru a třífázovým zapojením statoru. Obsahují zpětnou vazbu, která zajišťuje přesné nastavení polohy.

Krokové motory

Krokový motor je synchronní motor přeměňující vstupní digitální signál na mechanický pohyb. Umožňuje řídit otáčky i polohu rotoru. Nevýhodou je trvalý proudový odběr i v případě, kdy se motor netočí. Poměr hmotnosti ku kroutícímu momentu je horší než u stejnosměrných motorů. Pro běžné aplikace není nutné použít zpětnou vazbu. Krokové motory vyžadují stejnosměrné napájení. Jsou kompatibilní s digitálním řízením. Vždy vyžadují použití řídícího obvodu.

1.2 Konstrukce krokových motorů

Krokový motor se skládá ze statoru a rotoru. Stator je tvořen sadou cívek. Pólové nástavce statoru mají stejné vroubkování jako rozteč magnetů zubu rotoru. To umožňuje zvýšit přesnost při stejném počtu cívek. Rotor je tvořen hřídelí usazenou v ložisku. Tato hřídel je osazena permanentními magnety.

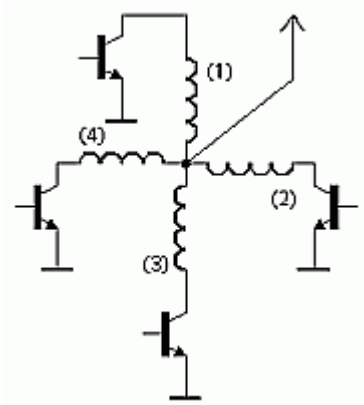
Princip krokového motoru: Proud procházející cívkou statoru vytvoří magnetické pole, které přitáhne opačný pól magnetu rotoru. Systematickým zapojováním cívek dosáhneme vzniku rotujícího magnetického pole. Vzniklé pole otáčí rotorem.

1.3 Řízení krokových motorů

Metodu řízení volíme podle požadovaného kroutícího momentu, přesnosti nastavení polohy a přípustného odběru. Rychlost otáčení motoru je omezena na několik stovek kroků za minutu kvůli přechodným magnetickým jevům. Rychlost bývá vyjádřena jako frekvence pulzů [Hz]. Při překročení maximální rychlosti začíná motor ztrácet kroky.

Unipolární řízení

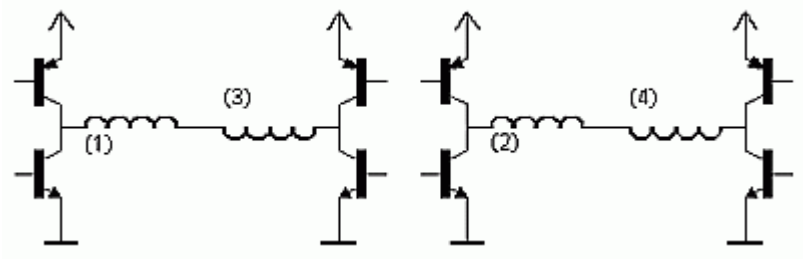
Při unipolárním řízení prochází proud v daném čase jen jednou cívkou. Motor má menší odběr a nižší kroutící moment než u bipolárního řízení. Řídící elektronika je jednodušší (spíná se vždy jen jedna cívka).



Obr. 1.1 Schéma principu unipolárního řízení [3]

Bipolární řízení

Při bipolárním řízení prochází proud v daném čase dvěma protilehlými cívkami. Cívky jsou zapojeny tak, aby vybudili navzájem opačné magnetické pole. Motor má vyšší odběr a vyšší kroutící moment než u unipolárního řízení. Řídící elektronika je složitější.



Obr. 1.2 Schéma principu bipolárního řízení [3]

Jednofázové řízení

Při jednofázovém řízení magnetické pole vytváří jedna cívka (u bipolárního buzení případně dvojice).

Dvoufázové řízení

Při dvoufázovém řízení magnetické pole vytváří dvě sousední cívky. Kroutící moment je vyšší než u jednofázového řízení. Spotřeba je dvojnásobná oproti jednofázovému řízení.

Plný krok (Full step)

Při řízení plným krokem je potřeba tolik kroků kolik má stator motoru magnetických zubů.

Poloviční krok (Half-step)

Pří řízení polovičním krokem je potřeba dvojnásobný počet kroků než u řízení plným krokem. Chod motoru je méně trhaný. Řízení probíhá pomocí střídání jednofázového a dvoufázového řízení.

Malý krok(Microstep)

Zde umožňuje řídící elektronika vytvořit menší úhel kroku než je dán počtem magnetických zubů. Tato metoda se používá u hybridních krokových motorů. Vyžaduje

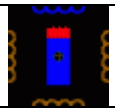
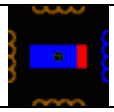
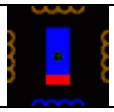
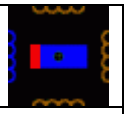
napájení kvazisinusovým průběhem místo obdélníkového. Také vyžadují upravenou konstrukci motoru [4].

1.4 Praktické řízení krokových motorů

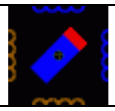
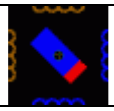
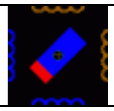
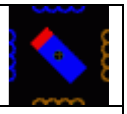
Cívka nakreslená hnědě (hodnota v tabulce označená "0"), je bez proudu. Magnetické pole modře nakreslené cívky (hodnota v tabulce označená "-") přitahuje červený konec magnetu (rotoru). Magnetické pole červeně nakreslené cívky (hodnota v tabulce označená "+") přitahuje modrý konec magnetu. Následující tabulky jsou uvedeny pro názornost.

Obr. 1.3 Fyzické rozmístění cívek [3]

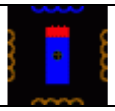
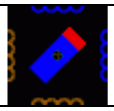
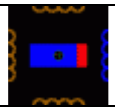
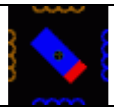
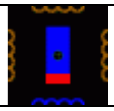
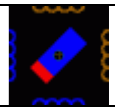
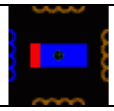
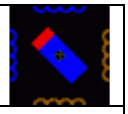
Tab. 1.1 Unipolární jednofázové řízení s plným krokem [3]

				
Cívka 1	-	0	0	0
Cívka 2	0	-	0	0
Cívka 3	0	0	-	0
Cívka 4	0	0	0	-

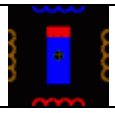
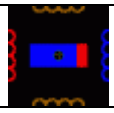
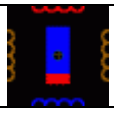
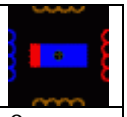
Tab. 1.2 Unipolární dvoufázové řízení s plným krokem [3]

				
Cívka 1	-	0	0	-
Cívka 2	-	-	0	0
Cívka 3	0	-	-	0
Cívka 4	0	0	-	-

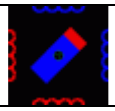
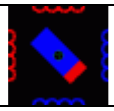
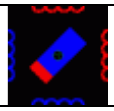
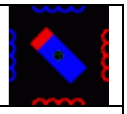
Tab. 1.3 Unipolární řízení s polovičním krokem [3]

								
Cívka 1	-	-	0	0	0	0	0	-
Cívka 2	0	-	-	-	0	0	0	0
Cívka 3	0	0	0	-	-	-	0	0
Cívka 4	0	0	0	0	0	-	-	-

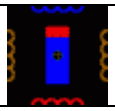
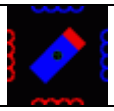
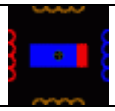
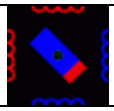
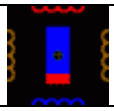
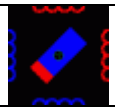
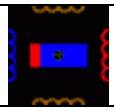
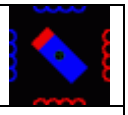
Tab. 1.4 Bipolární jednofázové řízení s plným krokem [3]

				
Cívka 1	-	0	+	0
Cívka 2	0	-	0	+
Cívka 3	+	0	-	0
Cívka 4	0	+	0	-

Tab. 1.5 Bipolární dvoufázové řízení s plným krokem [3]

				
Cívka 1	-	+	+	-
Cívka 2	-	-	+	+
Cívka 3	+	-	-	+
Cívka 4	+	+	-	-

Tab. 1.6 Bipolární řízení s polovičním krokem [3]

								
Cívka 1	-	-	0	+	+	+	0	-
Cívka 2	0	-	-	-	0	+	+	+
Cívka 3	+	+	0	-	-	-	0	+
Cívka 4	0	+	+	+	0	-	-	-

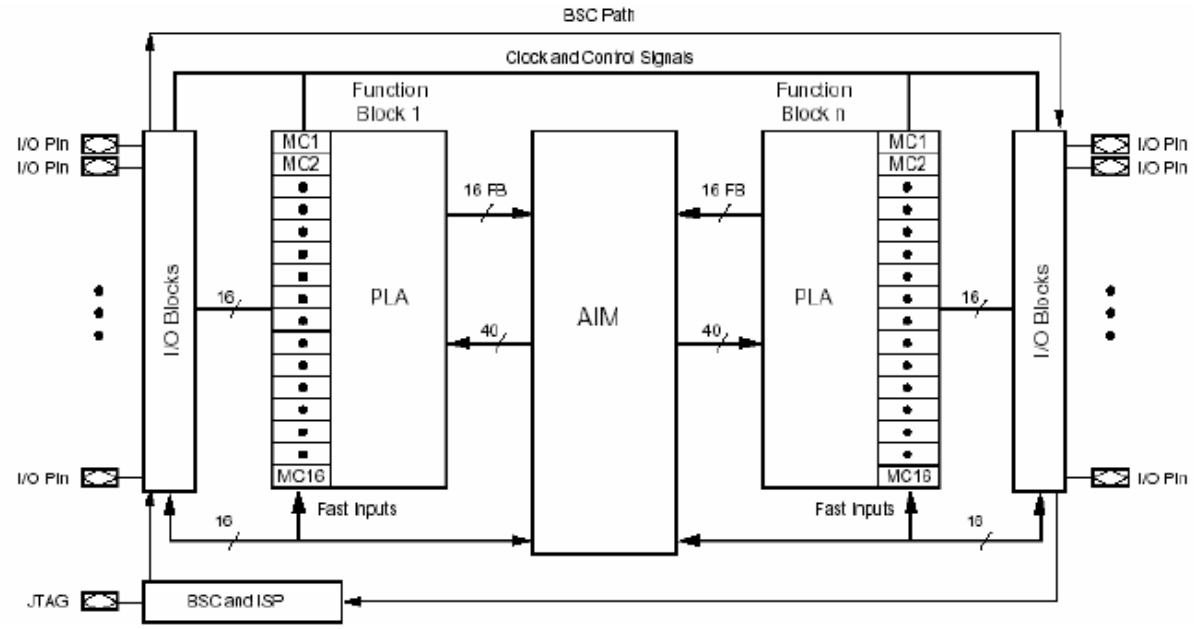
2. PROGRAMOVATELNÉ LOGICKÉ OBVODY

2.1 Popis obvodů PLD

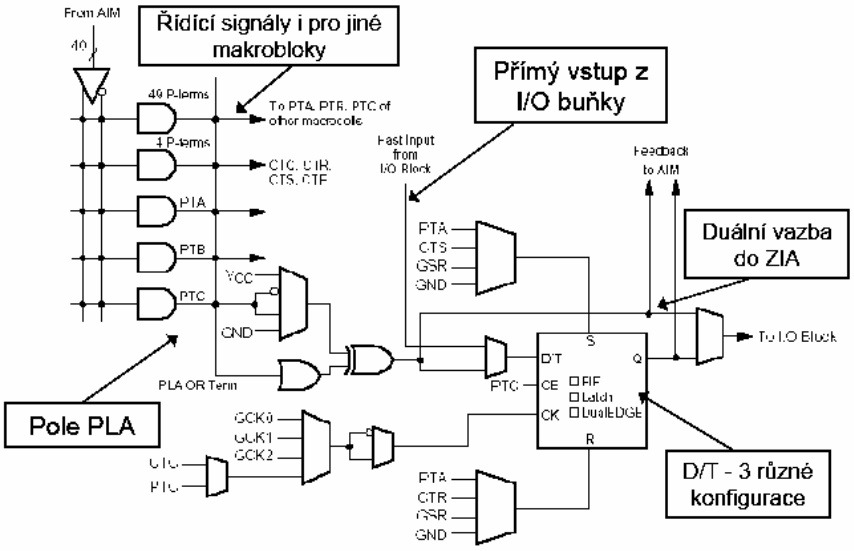
Zkratka PLD označuje Programmable Logic Device (Programovatelné Logické zařízení). Tyto obvody se používají k realizaci logických funkcí v jednom pouzdře. Logické funkce mohou být kombinačního i sekvenčního charakteru.

Realizace PLD je možná pomocí několika principů (paměťí PROM, PLA, PAL). V současné době jsou používány obvody SPLD, CPLD, FPGA. Obvody SPLD Simple Programmable Logic Device (Jednoduché Programovatelné Obvody) obsahují pevně definované vstupy a výstupy, mají malý počet hradel a klopných obvodů. Makrobuňky představují jednotlivé členy funkce ve tvaru součtu součinů. Tyto obvody mají použití pro velmi jednouché aplikace.

Obvody CPLD Complex Programmable Logic Device (Komplexní Programovatelné Logické Zařízení). Používají se pro malé aplikace, jejichž výhoda je zejména rychlost (řádově 100 MHz). Jejich architektura je tvořena centrálním propojovacím polem spojující jednotlivé logické bloky s výstupním propojovacím polem a vstupně/výstupními bloky, viz obr. 2.1. Logický blok je tvořen maticí mnohavstupových hradel AND a makrobuňkami. Uspořádání je znázorněno na obr. 2.2.



Obr. 2.1 Architektura obvodu Cool Runner 2 [5]



Obr. 2.2 Makrobuňka Cool Runner 2 a její připojení [5]

Obvody FPGA Field Programmable Gate Array (Programovatelná hradlová pole) jsou tvořeny pomocí programovatelných logických bloků (Logic Cell). Logické bloky jsou tvořeny generátorem logických funkcí pomocí pamětí (LUT). Tyto bloky se propojují pomocí globální propojovací matice. Obvody FPGA mohou navíc obsahovat další speciální bloky např. násobičky, paměti RAM, PLL a další. Nastavení obvodu je nutné obnovovat vždy po zapnutí ze vnitřní paměti RAM.

Programovatelné logické obvody se popisují pomocí schémat či programovacích jazyků. Pro jednoduché obvody se používá jazyk ABEL u složitějších VRILOG, VHDL, Systém C [5].

2.2 Vývojová deska XC2-XL

Vývojová deska slouží k vyzkoušení a odladění programu či jednotlivých komponentů konstrukce. V aplikaci je použita vývojová deska XC2-XL firmy Xilinx.

Jádrem vývojové desky jsou obvody Xilinx CoolRunner-II XC2C256 CPLD v pouzdře TQ144 a Xilinx XC9572XL CPLD v pouzdře VQ44. Na vývojové desce je umístěn konektor JTAG k oběma CPLD. JTAG port se používá k ladění a naprogramování obvodu z vývojového prostředí Xilinx ISE Design Suite 10.1. Napájecí část lze připojit k síťovému transformátoru, externímu zdroji či k baterii. Součástí napájecí části jsou stabilizátory napětí 3,3 V a 1,8 V. Deska obsahuje krystal pracující na frekvenci 1,8432MHz, s možností výměny za jiný do frekvence 100 MHz. I/O signály obvodů CPLD jsou spojeny s rozšiřujícími konektory. Vstupní a výstupní piny obvodu XC2C256 jsou spojeny s 40 pinovými výstupními konektory označenými A,B,D. Ke konektorům jsou také připojeny napájecí napětí VDD33 (3,3V), neregulované napětí U (5V) a zem GND. Obvod XC9572XL je připojen ke konektoru C. Pro základní signalizaci a obsluhu zařízení jsou přímo na desce umístěny dvě signalizační LED diody a tlačítko.

Navržený obvod je realizován v CoolRunner-II XC2C256, obsahující větší počet makrobuněk - 256. Hlavní charakteristiky použitého obvodu CoolRunner-II XC2C256 z literatury [5]: Obvod má statickou spotřebu menší než 100μA, označováno jako architektura FZP (FastZeroPower). Paměť E2CMOS vytvořená technologií 0.18 μm, 1000 P/E cyklů, životnost 20let. Napájecí napětí 1.8V, vstupně/výstupní obvody tolerují hodnoty napětí 1.5, 1.8, 2.5 a 3.3V.

Programování pomocí 1.8V ISP a JTAG (BoundaryScan) IEEE 1149.1. Klasická CPLD struktura s 32 až 512 MC v blocích 40V16. Bloky PAL využívají pole PLA 40x56x16 bez expandérů. Speciální obvody pro rozvod hodinového signálu – dělička 2,4,6,8,10,12,16, řízení registrů náběžnou i sestupnou hranou (DualEDGE), CoolCLOCK. Možnost odpojení vstupů pro snížení spotřeby – DataGATE. Banky I/O buněk s odlišným napájením - od 128 MC.
I/O buňky je možné vybavit Schmittovými klopnými obvody, pull-up a funkcí bus-hold.

Vývojová deska obsahuje jedno tlačítko a dvě signalizační LED diody. Pro účely zkoušení a ověření funkčnosti je použita rozšiřující deska DIO4 firmy Digilent. Rozšiřující deska se připojuje pomocí dvou 40 pinových konektorů na porty A a B vývojové desky. K napájení rozšiřující desky je potřeba napětí 3,3V. Pro PS/2 konektor pak i napětí 5V vytvořené pomocí stabilizátoru na vývojové desce s nízkým úbytkem.

Rozšiřující deska obsahuje 4 místný sedmi-segmentový LED displej se společnou anodou. Displej je přímo připojen k 40 pinovému konektoru. K zobrazení se využívá dynamické řízení. Deska dále obsahuje: 8 nezávislých LED diod. Data pro zobrazení LED diodami jsou uložena do záchytného klopného obvodu. 4 tlačítka vybaveny RC článkem a Schmittovým klopným obvodem. Tlačítka jsou odolná proti zákmitům. 8 spínačů bez zvláštních opatření. 3 bitový VGA port. PS/2 konektor pro myš a klávesnici.

2.3 Programovací jazyk VHDL

K programování obvodů CPLD je nejčastěji využíván jazyk VHDL. Pojem VHDL je akronymem zkratky VHSIC HDL. Její význam je Very High Speed Integrated Circuits Hardware Description Language. (Velmi rychlé integrované obvody Jazyk popisující hardware). Jazyk byl přijat jako standart IEEE 1076 v roce 1987. Nyní je aktuální standart z roku 2002 IEEE Std. 1076-2002 [7].

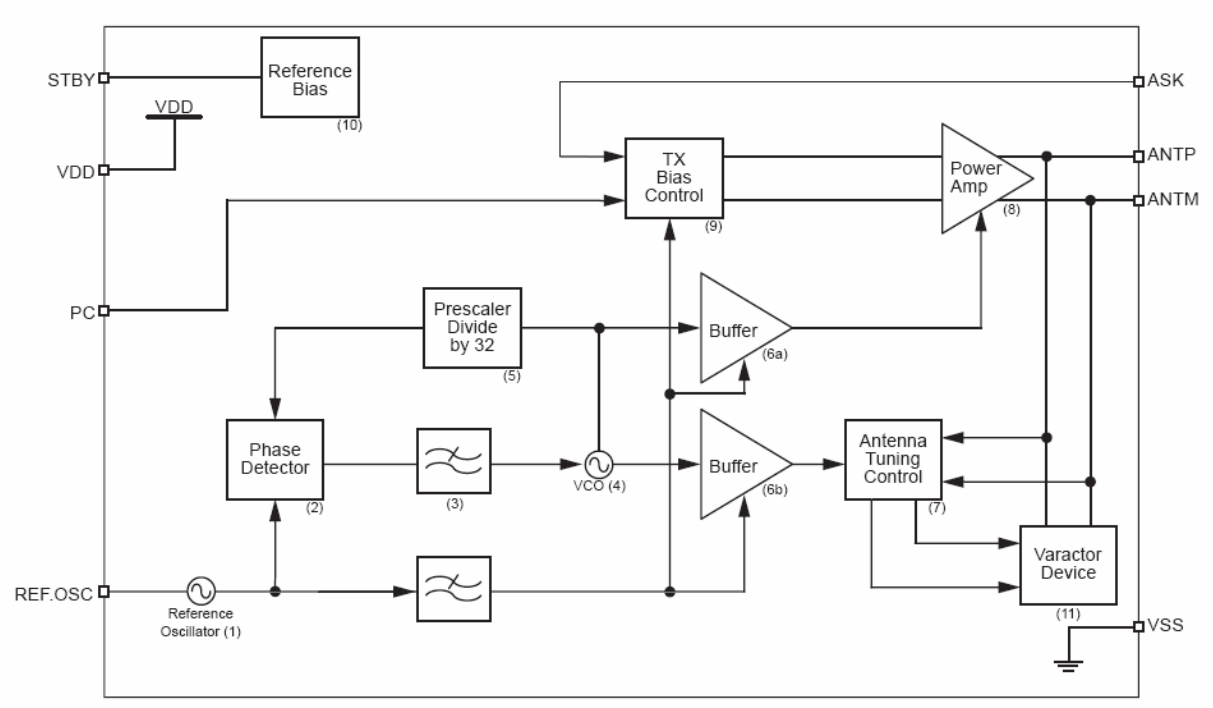
Jazyk VHDL je otevřený standart (k použití není potřeba licence). Konkrétní typ cílového obvodu je vhodné zvolit až po vytvoření programu (nezávislost vývoje). Umožňuje návrh jak sekvenčních tak kombinačních obvodů. Zdrojový kód je možné dále použít pro jiné simulace, aplikace (přenositelnost kódu). Mezi nevýhody jazyka VHDL patří možné vytvoření neefektivní konstrukce, která je závislá na zkušenostech konstruktéra a návrhovém systému (syntezátoru). Jazyk VHDL byl původně vytvořen pro modelování a simulaci rozsáhlých systémů [6].

3. DOPLŇUJÍCÍ OBVODY

3.1 Radiová část

Radiová část slouží k bezdrátovému přenesení informace mezi vysílačem Tx a přijímačem Rx. Komunikace postačuje jedním směrem. Není vyžadována informace o potvrzení přijmutí signálu. Radiové obvody pracují v pásmu 433 MHz. Využívání tohoto pásma je vymezeno všeobecným oprávněním VO-R/10/03.2007-4 [12]. Dosah použitých obvodů je několika metrů při přímé viditelnosti.

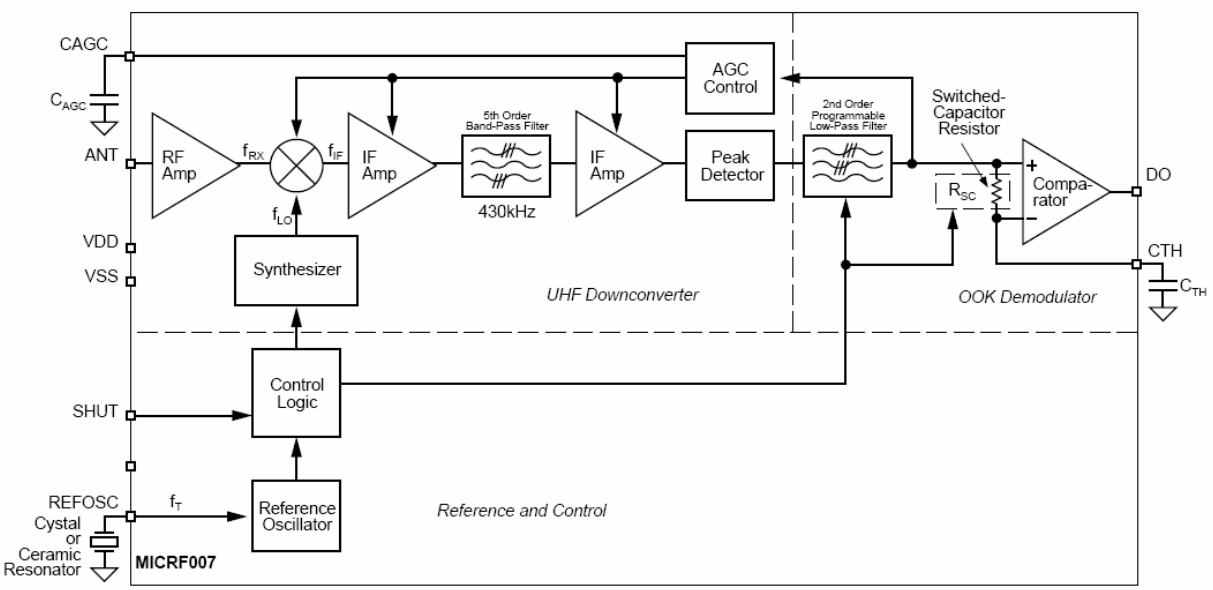
Radiový vysílač je navržen pomocí integrovaného obvodu MICRF102. Tento obvod v sobě sdružuje kompletní UHF vysílač s frekvenčním rozsahem 300 MHz až 470 MHz. Obvod používá klíčování ASK (Amplitude Shift Keying) s modulační rychlostí 0,1 až 20 Kb/s. Blokové schéma obvodu MICRF102 je na obr. 3.1. Bloky označené (1,2,3,4,5) jsou součástí UHF syntetizátoru se smyčkou fázového závěsu vytvářejícím nosnou frekvenci s kvadraturní modulací. Soufázový signál I je použit na řízení výkonového zesilovače (8) přes zesilovač (6a). Kvadraturní složka Q je přes zesilovač (6b) použita k dolaďování rezonančního obvodu. Obvod automaticky dolaďuje frekvenci sledováním fáze na výstupu výkonového zesilovače (8). Paralelně s rezonančním obvodem je zapojen varikap (11) řízený stejnosměrným napětím bloku anténního dolaďování (7). Vlastní modulace je prováděna obvodem kontrolujícím přepětí výkonového zesilovače (9).



Obr. 3.1 Blokové schéma vysílacího obvodu MICRF102 [9]

Radiový přijímač je navržen pomocí integrovaného obvodu MICRF007. Tento obvod v sobě sdružuje kompletní UHF přijímač s frekvenčním rozsahem 300 MHz až 440 Mhz. Obvod automaticky dolaďuje frekvenci smyčkou fázového závěsu. Rychlost demodulovaného signálu do 3200 bitů/s. Obvod je kompatibilní s logickými úrovněmi TTL. To usnadňuje jeho připojení k logickým obvodům zpracovávajícím přijatý signál. Blokové schéma je na obr. 3.2.

Demodulátor UHF (blok UHF Downconvetor) je typu superheterodyn s použitým úzkým filtrem. Střední frekvence f_T vytvořená rezonátorem je udržována syntezátorem (blok Synthetizer) se smyčkou fázového závěsu. Signál přijatý anténou je přiveden do vysokofrekvenčního zesilovače (blok RF Amp) a směšovače. Signál je poté zesílen (blok IF Amp) a jsou odfiltrovány nežádoucí kmitočtové složky (filtr 430 KHz). Blok AGC (Automatický řízení zesílení) upravuje velikost zesílení v závislosti na velikosti demodulovaného signálu. Signál z bloku UHF demodulátoru přichází přes dolní propust do komparátoru. Komparátor v závislosti na referenční hodnotě vytváří výstupní data.



Obr. 3.2 Blokové schéma přijímacího obvodu MICRF007 [10]

3.2 Snímání polohy

Snímání polohy je zpětná vazba k řídícímu obvodu, která přenáší informaci o pohybu. U krokového motoru může dojít vlivem velkého zatížení, mechanického selhání k přeskočení kroků a tím k nenastavení na požadovanou polohu. Běžně dostupné snímače pro snímání úhlové polohy můžeme rozdělit na absolutní – udávají informaci o přesné poloze a snímače inkrementální – ty udávají informaci o relativní změně polohy.

Nejvhodnější by bylo použít absolutní snímač polohy. Ten je však finančně nákladný. Inkrementální snímače jsou k dispozici v provedení optickém a mechanickém. Ze snímačů inkrementálních je finančně nákladný i optický snímač. Zbývá použít jen snímač mechanický. Rozlišení úhlové polohy je u mechanických snímačů do 30 impulsů na otáčku. Snímač je tedy schopný rozlišit 12˚. Krokový motor má však jeden krok o velikosti 1,8˚. Použití mechanického snímače není nejvhodnější.

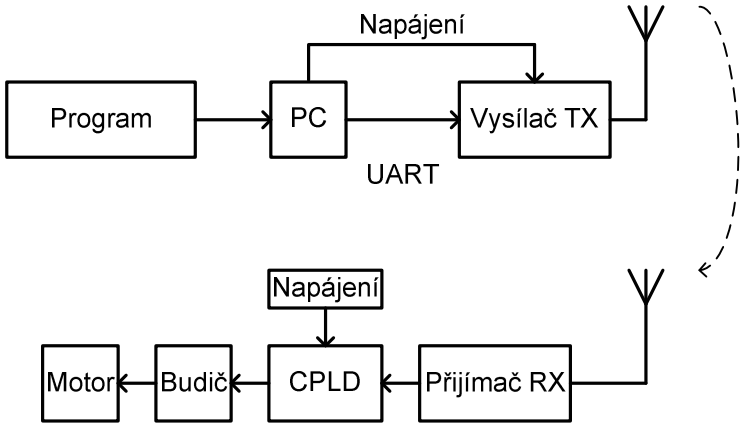
Pokud nejsou krokové motory přetěžovány není nutné je vybavit obvodem zpětné vazby [3]. Rozhodl jsem se tedy použít optickou závoru, která bude umístěna v pevné pozici. Pomocí optické závory se jednou za otočení o 360˚ zkontroluje poloha. Je-li jiná, došlo ke ztrátě kroku a pozice se upraví na správnou hodnotu. Nevýhodou řešení s optickou závorou je její mechanické provedení.

3.3 Uchování pozice

Aktuální poloha rotátoru je uložena v registru `cnt_reg`. Po vypnutí napájení vývojové desky dojde k vymazání registru a ke ztrátě aktuální polohy. Tento problém lze vyřešit automatickým nastavováním např. do nulové polohy hned po zapnutí. Toto řešení je zdlouhavé. Další možností je použití externí paměti pro uložení pozice. Paměť musí být typu nonvolatile, aby uchovala hodnoty bez napájecího napětí. Velikost paměti je dána rozsahem kroků motoru. Pro použitý motor je velikost paměti 8 bitů. Vhodné paměti z nabídky prodejců mají o mnoho větší velikost a cenu. Větší velikost pamětí je doprovázena složitějším adresovacím postupem a tím složitějším řídícím obvodem. Vývojová deska XC2-XL umožňuje využít napájení z baterií. Toto řešení je nejvhodnější.

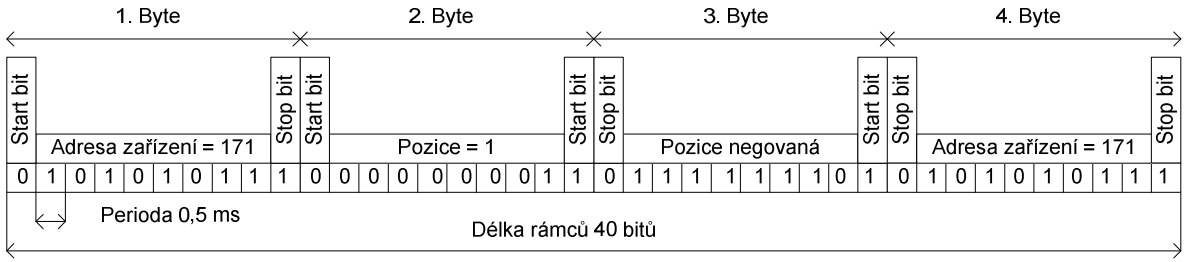
4. REALIZACE

4.1 Popis způsobu ovládání



Obr. 4.1 Způsob ovládání

Způsob ovládání je znázorněn na obrázku 4.1. Přenášená data představují úhlovou polohu 0 – 360˚ s rozlišením kroků krokového motoru. V PC běží program, který vysílá data o poloze kódovaná pro přenos. K PC je pomocí sériového rozhranní RS232 (com portu) připojen vysílač. Vysílač vyžaduje napájení 5V. To je připojeno z PC nebo z externího zdroje. Vysílač přenáší data okolním prostředím k přijímači.



Obr. 4.2 Formátování rámce

Vysílaná data jsou formátována do rámce podle Obr 4.2. Formát je kompatibilní s rozhranním UART. Pro asynchronní přenos jednoho bajtu používá jeden start bit, 8 datových bitů , jeden stop bit, paritní bit není použit. Přenáší se 4 bajty. První a poslední obsahuje zvolenou konstantu - adresu zařízení. Druhý bajt obsahuje pozici v rozsahu 0 – 199. Rozsah pozice odpovídá rozlišení kroků krokového motoru. Krokový motor má 200 kroků pro otočení o 360˚. Jeden krok je tedy 1,8 ˚. Třetí bajt obsahuje pozici jejíž binární hodnota je negována. V přijímači po přijetí celého rámce zkontroluje obvod CPLD vzájemnou negaci pozice v druhém a třetím bajtu. Zkontroluje také pevně nastavené hodnoty v prvním a čtvrtém bajtu spolu s úrovní všech start a stop bitů. Délka rámců 4 bajty je zvolena s ohledem na rušení z okolního prostředí. Jestliže vysílač vysílá, dochází k příjmu téměř bez chyb. Jestliže přestane vysílač vysílat je okolní šum a rušení přijímačem vyhodnocováno jako přijímaný signál. Formátování dat spolu s ověřením negace bajtů mají za cíl odstranit příjem náhodného signálu z okolí přijímače.

Přijímač přijme data vyhovující formátování. Obvod CPLD data dekóduje a určí novou polohu, na kterou se má rotátor natočit. Podle informace o aktuální poloze rotátoru

rozhodne o směru točení a počtu kroků motoru. Obvod CPLD řídí buzení jednotlivých cívek krokového motoru, aby se motor točil požadovaným směrem.

4.2 PC program

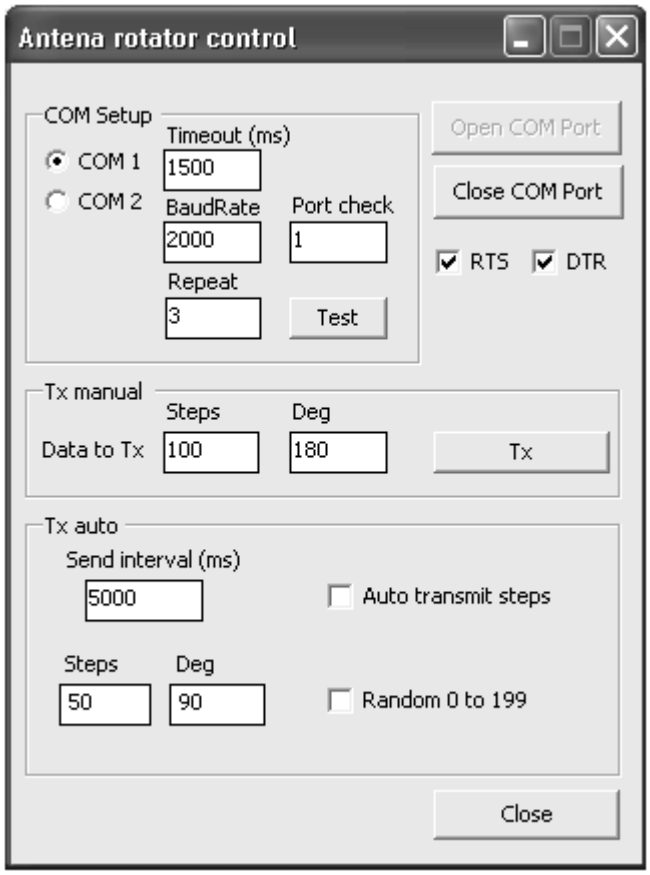
Program, běžící na osobním počítači, slouží ke kódování a vysílání dat. Data musí být zformátována do tvaru podle Obr 4.2. Jinak nedojde k jejich přijetí. Program komunikuje s rozhranním počítače RS232 podporující sériovou asynchronní komunikaci. Program může být libovolný umožňující komunikaci pomocí asynchronní komunikace rychlostí 2000 Bd s 8 dat. bity , jedním start a stop bitem. Avšak pro testovací účely jsem vytvořil program vlastní.

Program je napsán v jazyku Visual Basic 6.0 v prostředí Visual Studio. Jako základ programu jsem využil projekt [8]. Z něj jsem ponechal samotnou komunikaci s rozhraním RS232 a program doplnil o vlastní funkce a testovací sekvence.

Zdrojový kód části programu je uveden v příloze C. Zde je uvedena část programu která bezprostředně souvisí s formátováním a odesláním dat. Funkce `Button1_Click_1` vytvoří ze vstupní hodnoty datový rámec podle Obr 4.2. Tento rámec několikrát odešle na výstupní port PC. Počet opakování rámce je nastaven v textboxu repeat. Opakování rámce je zde kvůli okolnímu rušení. Na přijímací straně nedochází k opravám chyb pouze k detekci bezchybného příjmu. Komunikace je jednosměrná, přijímací zařízení nemůže sdělit vysílači jestli data přijal nebo ne. Několikanásobným opakováním datového rámce zajistíme správnou funkci i v zarušeném prostředí. Funkce `Timer1_Tick` slouží k automatickému odesílání rámců vždy po stanovené době.

Uživatelské rozhranní programu je znázorněno na Obr. 4.3. Program se připojí na port RS232 vybráním příslušného portu (COM1 nebo COM2) a stisknutím tlačítka Open COM port. V programu lze nastavit rychlost sériového přenosu Baudrate, počet opakování rámců Repeat a čas opakování Timeout. Tlačítko Test slouží k otestování připojeného COM portu. Checkboxy RTS (Request To Send) a DTR (Data Terminal Ready) při zaškrtnutí vysílají spolu se sériovou komunikací potvrzující signály na pinech 3 a 8 konektoru RS232. Tyto signály se dají využít pro napájení vysílací části.

Program je rozdělen do dvou sekcí. Sekce Tx manual vyšle rámec s pozicí danou hodnotou v textboxu. Hodnota může být zadána buď v krocích motoru nebo ve stupních. Obě hodnoty se hned po vyplnění vzájemně přepočítávají. Sekce Tx auto odesílá rámec automaticky v zadaném časovém intervalu Send interval (ms). Hodnoty pozice k odeslání jsou vytvořeny po krocích v textboxu Steps popřípadě stupních Deg. Např pro Obr. 4.3 budou po 5 vteřinách vyslány postupně hodnoty 90˚, 180˚, 270˚, 0˚, 90˚ . Podle nastavení jsou vyslány pozice v násobcích zadaného čísla nebo pozice náhodné. Automatický způsob zasílání se zapíná pomocí položky Auto transmit steps. Současným zapnutím položky Random 0 to 199 se budou v časovém intervalu odesílat náhodné hodnoty. Současně s odesláním se hodnoty v polích Steps a Deg aktualizují. Program se ukončí tlačítkem Close.



Obr. 4.3 Program RS232

4.3 Obvodové zapojení

Obvodové zapojení se skládá z vysílací části a části přijímací. Schéma zapojení vysílací části je v příloze A.1. Vysílací část je konektorem X1 připojena na port osobního počítače. Sériový signál je přiveden do převodníku úrovní RS232 – TTL označeného IC2. Použit je běžný převodník pro sériovou komunikaci MAX233. U tohoto obvodu je využita jen část Rx. Vhodnější by bylo použít obvod MAX3180, který obsahuje jen jednosměrný převodník z RS232 do TTL. Modulační signál je přiveden na klíčovací vstup vysílače IC1. Zapojení vysílacího obvodu IC1 MICRF102 odpovídá katalogovému zapojení [9]. K vysílacímu obvodu je připojen rezonanční obvod se smyčkovou anténou tvořený součástkami C2, C3, L1. Kondenzátory C6, C5, C4 filtrují napájecí napětí. Odporový dělič R1, R2 spolu s C1 udržují provozní úroveň napětí na vstupu power control integrovaného

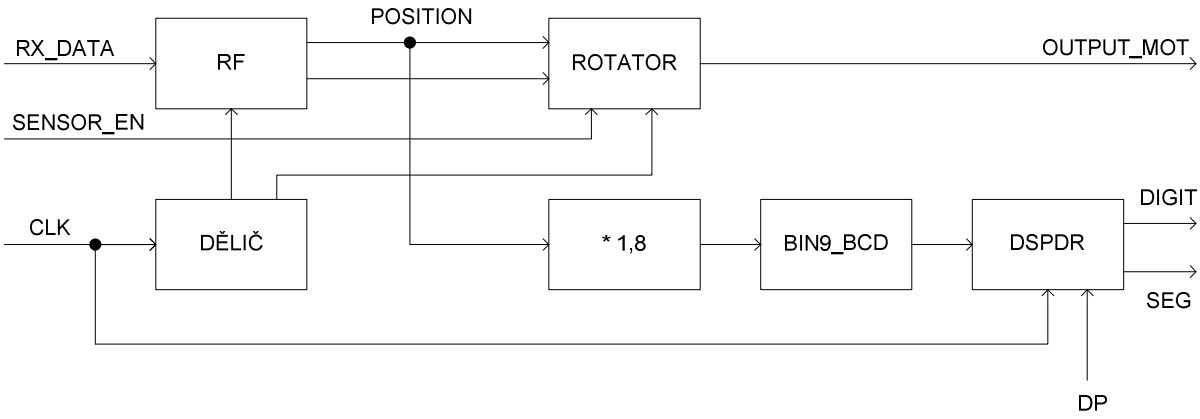
obvodu IC1. Připojený krystalový oscilátor má frekvenci 32-krát menší než je vysílací frekvence obvodu. Po konzultaci s vedoucím, kvůli nedostupnosti krystalů, byla zvolena vysílací frekvence 327,84 MHz. Integrované obvody IC1 a IC2 jsou napájeny přes filtrační kondenzátory C7, C8, C9. Napájecí napětí je připojeno buď z linky RS232 přes usměrňovací diody D1, D2 stabilizováno zenerovou diodou D3 nebo z externího zdroje napětí o hodnotě 5V připojeného na konektor JP2. Napájení z linky RS232 se zapíná či vypíná propojkou JP3. Propojka JP1 umožňuje vypnou funkci vysílacího obvodu IC1. Navržená oboustranná deska plošných spolů je umístěna v příloze A.2 a A.3. Seznam součástek pak v A.4.

Schéma zapojení přijímací části je v příloze B.1. Přijímací část je konektorem SV1 připojena k vývojové desce s obvodem CPLD. Tento obvod zařizuje vlastní zpracování dat a řízení krokového motoru. Radiový přijímací obvod IC1 MICRF007 přijímá signál zachycený anténou ANT. Signál demoduluje a předá ke zpracování odvodu CPLD. Obvod IC1 je zapojen podle doporučeného katalogového zapojení [10]. K odfiltrování nežádoucích vysokofrekvenčních složek je zapojena tlumivka L1 a kondenzátor C1. Obvod IC1 je napájen napětím 5V z vývojové desky XC2-XL. K filtraci napájení IC1 jsou použity kondenzátory C4, C5, C6. Kondenzátor C3 je nutný kvůli automatickému dolaďování zesílení obvodu. Na kondenzátoru C2 je referenční hodnota napětí pro vnitřní komparátor. K obvodu je připojen krystal Q1 s frekvencí 64,5-krát menší než je vysílací frekvence obvodu. Obvod IC2 ULN2803 obsahuje 8 spínacích tranzistorů spolu s ochrannými diodami v jednom pouzdře. Slouží ke spínání jednotlivých vinutí krokového motoru připojeného ke konektoru SV2. K SV2 je připojen unipolární krokový motor s impedancí cívek 75 Ω. Pro hodnotu napájecího napětí 12V vychází proud spínacím tranzistorem a jednou cívkou 160 mA. Obvod IC2 je dimenzován na 500mA a není ho nutné chladit. Napětí 12V pro spínání cívek je připojeno přes konektor K1.

4.4 Řešení řídícího obvodu

Program je napsán v programovacím jazyku VHDL v návrhovém systému Xilinx ISE Design Suite 10.1. Je naprogramován do vývojové desky XC2-XL do obvodu Xilinx CoolRunner-II XC2C256.

Program se skládá z několika částí spojených v hlavním modulu *`main.vhd`*. Zdrojový kód je uveden v příloze D.1. Hlavní modul propojuje signály nižších modulů podle obr 4.4.



Obr. 4.4 Blokové zapojení modulů CPLD

Výstupními signály hlavního modulu jsou `output_mot` připojené k budiči cívek motoru, `rx_out` připojené k LED diodám na rozšiřující desce a signály `digit4` a `seg`

připojeny k sedmi-segmentovému displeji na rozšiřující desce. Vstupními signály je hodinový signál `clk` s frekvencí 1,842 MHz z krystalového oscilátoru vývojové desky, `rx_data` signál z přijímače a `sensor_en`, představující signál z optické závory prozatím umístěn na tlačítko rozšiřující desky. Kromě propojení hlavní modul vytváří hodinové signály ze vstupního signálu `clk`. Proces na řádku 90 vytváří pomocí děličů `clk_div` a `cnt_32k` hodinový signál pro přijímací obvod 32 KHz a hodinový signál pro krokový motor 112 Hz. Na řádku 107 je výpočet binární pozice ve stupních. Výpočet je realizován pomocí vztahu position * 231/128. Zlomek `231/128` představuje velikost kroku krokového motoru s hodnotou 1,8. Tento převod je použit pro zobrazení na displeji. Přesnost čísla 1,804 je dostatečná s odchylkou ±1 na displeji.

Část vytvářející dekódovací obvod přijímače je umístěna v modulu *`rf.vhd`* jeho zdrojový kód je uveden v příloze D.2. Vstupními signály modulu jsou sériový demodulovaný signál `rx_data` a hodinový signál `rx_16clk` s frekvencí 16krát větší, než je frekvence modulačního signálu. Výstupem modulu je 8 bitové datové slovo s pozicí `rx_out` a signál potvrzující přijetí dat `data_ready`. Proces začínající na řádku 35 zjišťuje, jestli nedošlo ke změně vzorků signálu `rx_data`. Jestli ano, nastaví signál `edge = 1`. Proces na řádku 46 vytváří hodinový signál pro přijímač. Je odebírán z nejvyššího bitu děliče `clkdiv`. Hodinový signál `rx_clk` se upraví, synchronizuje s každou náběžnou nebo sestupnou hranou signálu `rx_data`. Aby k synchronizaci `rx_clk` nedocházelo na základě přijatých rušivých signálů zjišťují se vzorky pro porovnání s 16-ti násobnou frekvencí hodinového signálu. Proces na řádku 59 odebírá vzorky signálu `rx_data`. Platný vzorek datového rámce se odebírá v polovině bitové periody signálu `rx_data`. Registr `regrx` tvoří datový rámec, do kterého se sériově ukládají vstupní data. V registru `regrx` se zkontrolují start bity, stop bity a vzájemná negace dat o poloze. Po splnění této podmínky jsou data zapsána na výstup `rx_out`.

Řízení otáčení krokového motoru na pozici určenou signálem `position` je umístěno v modulu *`rotator.vhd`* jeho zdrojový kód je uveden v příloze D.3. Výstupními signály modulu jsou signály `output_mot` ke spínání cívek motoru. Cívky jsou spínány přes přímý budič úrovní log. 1. Vstupními signály modulu jsou hodinový signál `clk`, údaj o pozici signál `position`, spouštěcí signál `position_en` a `sensor_en`. Spouštěcí signál `position_en` svou náběžnou hranou spustí přijetí pozice a roztočení krokového motoru. Spouštěcí signál `sensor_en` slouží k upravení pozice vnějším senzorem. Konstanta `StepsMax` určuje počet kroků krokového motoru pro otočení o 360˚. Použitý krokový motor má 200 kroků na otáčku, jedním krokem se otočí o 1,8˚. Při výměně krokového motoru nebo přidání převodovky je nutné změnit konstantu `StepsMax` a případně bitové šířky navazujících signálů. Konstanta `Sensor_position` obsahuje údaj o poloze optické závory v krocích. Proces na řádku 44 uloží přijatou pozici do signálu `cnt_catch`. Určí z aktuální a nové pozice směr otáčení pro nejkratší cestu k nové pozici. Proces na řádku 84 upravuje hodnotu registru s aktuální pozicí `cnt_reg`. Je-li tento registr shodný s novou pozicí ukončí otáčení krokového motoru. Proces také řídí upravení pozice signálem `sensor_en`. Tento vstupní signál je veden např. z optické závory umístěné v pevné pozici (úhlu). Dojde-li mechanickým či jiným vlivem k přeskočení několika kroků motoru. Při průchodu optickou závorou signál `sensor_en` uloží do registru s aktuální polohou `cnt_reg` polohu umístění optické závory `Sensor_position`. Tím dojde k opravení chybné polohy. Není-li poloha chybná pozice v registru `cnt_reg` je shodná s hodnotou umístění optické závory.

Obvod pro samotné otáčení pracuje na principu stavového automatu Mealyho typu. Stavový konečný automat je zařízení, které v přesně definovaných okamžicích přechází mezi definovanými stavy. Další stav do kterého automat přejde je určen současným stavem a stavovými signály. Stavové automaty mohou být Mealyho a Moorova typu. U automatů Moorova typu dochází ke změně stavu při změně stavové veličiny, výstupní hodnota závisí jen na stavu automatu. U automatů Mealyho typu změna nastává až při synchronizaci hodinovým signálem, výstupní hodnota závisí na stavu automatu a vstupních signálech.

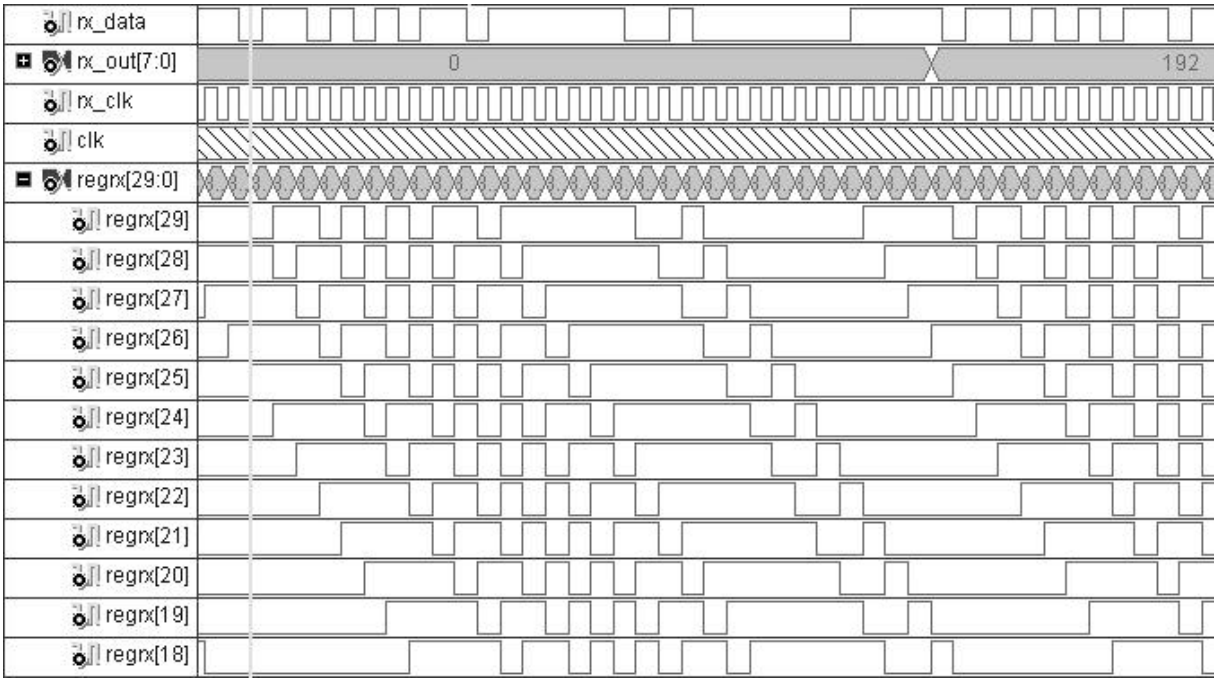
Stavový automat se skládá z kombinační části a části sekvenční. Sekvenční část je reprezentována procesem začínajícím na řádku 84. V této části dochází k zapsání přichystaných hodnot pro nový stav `to_out` na výstup obvodu `output_mot`. Zápis je proveden jen s náběžnou hranou hodinového signálu `clk`. Část kombinační je reprezentována procesem začínajícím na řádku 120. Proces je spuštěn při změně aktuálního stavu `now_ST` nebo vstupních signálů `full_half` a `forward_backward`. Proces nastavuje další stav automatu v závislosti na řídících signálech `full_half` a `forward_backward` a připravuje výstupní hodnoty pro budoucí stav. Pro unipolární krokový motor jsou hodnoty z tabulky 2.3 přiřazené proměnné `to_out`. Aktivní hodnota signálu je log. 1. Vstupní signál obvodu `full_half` určuje jestli se krokový motor otáčí plným krokem `full_half = 1`, nebo polovičním krokem `full_half = 0`. Vstupní signál obvodu `forward_backward` určuje jestli se krokový motor točí vpřed `forward_backward = 1`, nebo vzad `forward_backward = 0`. Stavový automat obsahuje vstupní signál `run`. Při `run = 0` je řídící obvod neaktivní a při `run = 1` řídící obvod pracuje.

Modul *`DspDr.vhd`* zobrazuje na sedmi-segmentovém displeji rozšiřující desky údaj přijaté polohy ve stupních. Jeho zdrojový text je umístěn v příloze D.4. Vstupními signály modulu jsou hodinový signál `clk`, 12ti bitové číslo v BCD kódu `bcdint`. Signály `dp3,dp2,dp1,dp0` pro zobrazení desetinné tečky nejsou využity. Výstupními signály jsou pozice displeje `digit` a signál pro sedmi-segmentový displej `seg`. Signály `digit` a `seg` jsou připojeny pomocí rozšiřující desky přímo k led diodám displeje. Displej je řízen dynamicky. Střídavě několikrát za vteřinu jsou poslány hodnoty `seg` na jednotlivá čísla displeje. Přepínací signál 1 kHz pro displej vytváří procesy na řádcích 24 a 36. V procesu na řádku 52 dochází z vytvoření signálu `seg` a `digit`. Signál `seg` je výstupem kodéru z kódu BCD na kód sedmi-segmentového displeje. Využity jsou jen dekadická čísla 0 až 9 a speciální znak stupně.

Modul *`bin9_bcd.vhd`* realizuje devítibitový paralelní převodník kódu binárního do kódu BCD, jeho zdrojové schéma je v příloze D.5. Převod je vyžadován modulem *`DspDr.vhd`* který zobrazuje data z BCD kódu. Převodník je realizován pomocí sčítačky `add3`, její zdrojový kód je umístěn v příloze D.6. Repliky sčítačky jsou postupně pospojovány podle [11]. Pospojování je zapsáno schématicky z důvodů přehlednosti.

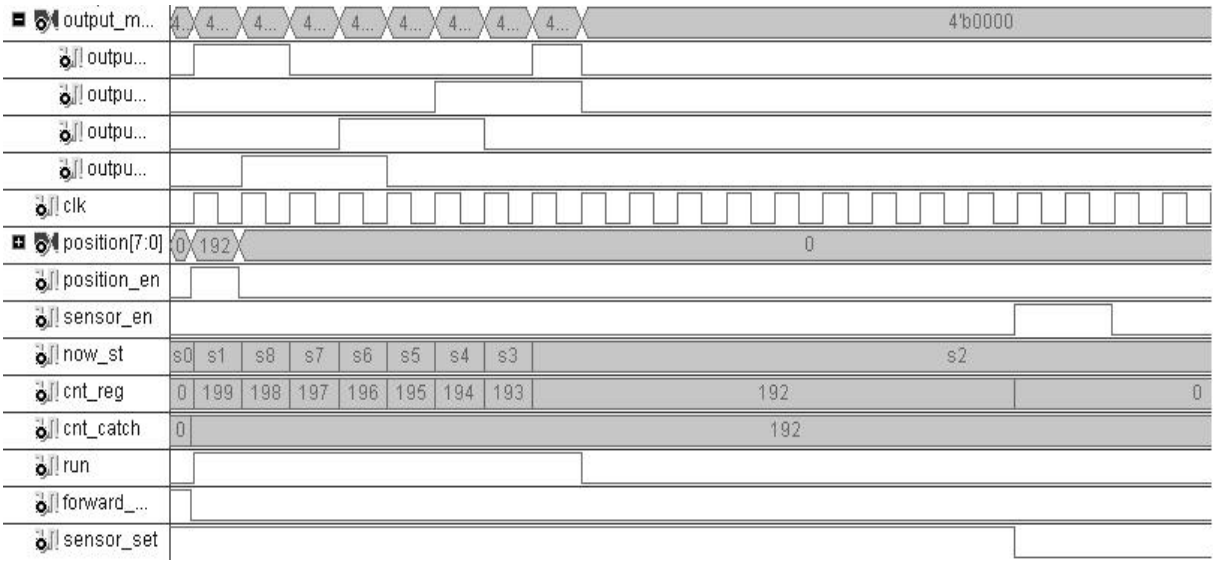
4.5 Simulace průběhů řídícího obvodu

Simulace jsou provedeny v programu Xilinx ISE Design Suite 10.1. Na obr 4.5 jsou znázorněny průběhy signálů modulu `rf.vhd`. K odebírání vzorku vstupního signálu dochází s náběžnou hranou signálu `rx_clk` v polovině bitové periody signálu `rx_data`. Data jsou sériově ukládána do registru `regrx`. Po splnění tvaru datového rámce jsou přijatá data uloženy na výstup `rx_out`.



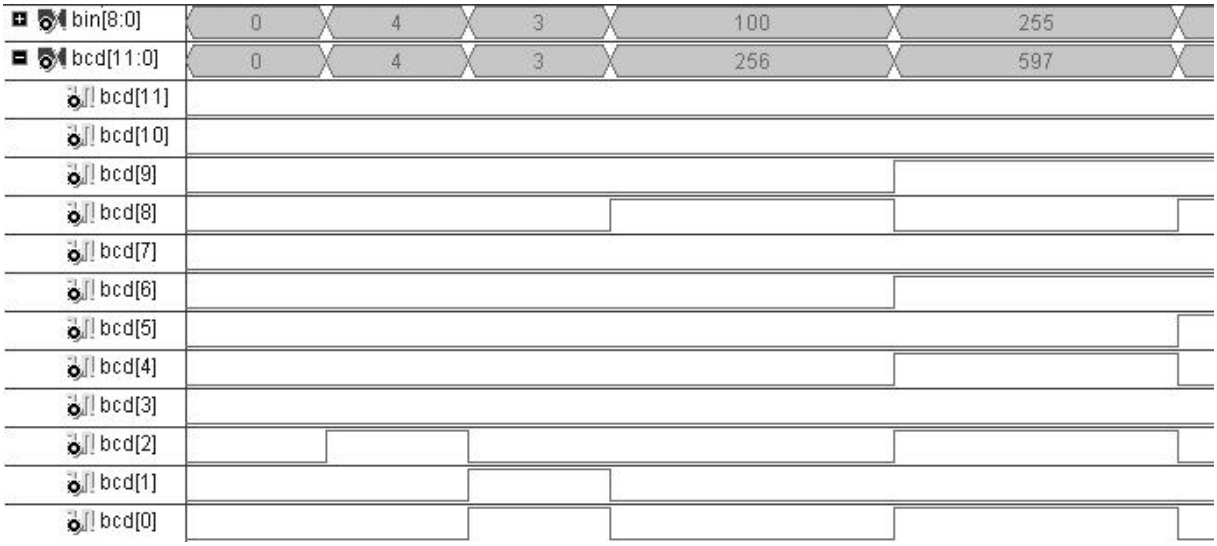
Obr. 4.5 Simulační průběhy modulu RF

Na obr 4.6 jsou znázorněny průběhy signálů modulu `rotator.vhd`. Stav stavového automatu se mění s hodinový signálem `clk`. Aktuální stav je uložen v signálu `now_st`. Každému stavu odpovídají výstupní hodnoty `output_mot`. Při přijetí signálu `position` signálem `position_en` se umístí hodnota pozice do registru `cnt_catch`. Určí se směr otáčení `forward_backward` a řídícím signálem `run` otáčení začne. Aktuální hodnota pozice je umístěna v registru `cnt_reg`. Při shodě registru `cnt_reg` a `cnt_catch` se stavový automat zastaví. Signál `sensor_en` aktivní v log. 1 vnutí do `cnt_reg` hodnotu pozice senzoru v tomto případě 0. Tím dojde k opravení chybné pozice např. při vynechání kroku motoru.



Obr. 4.6 Simulační průběhy modulu Rotator

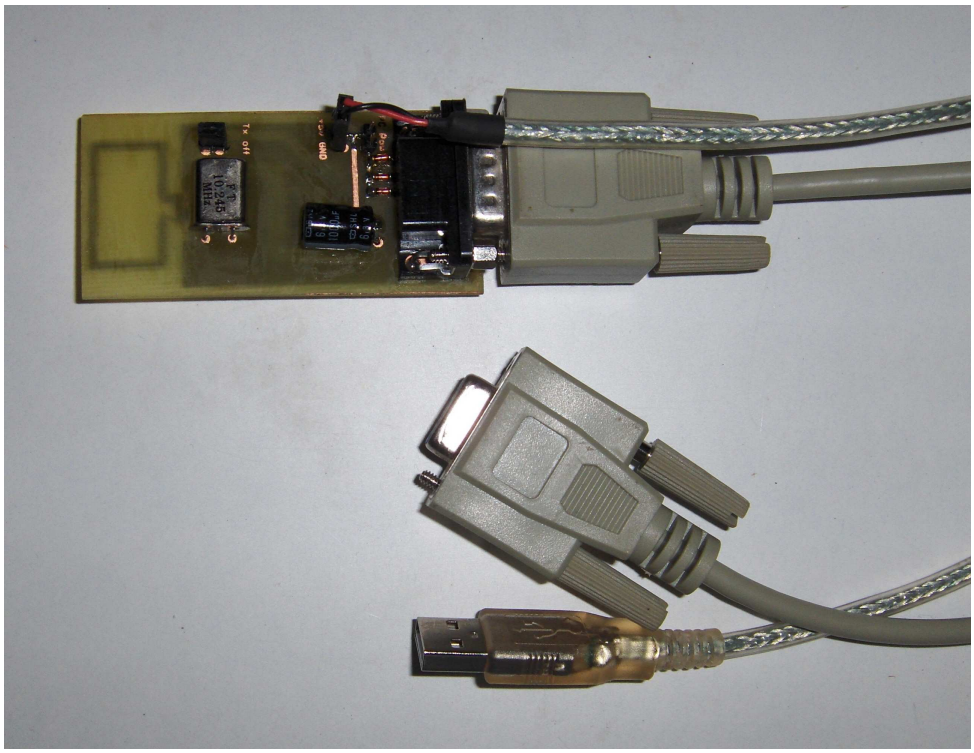
Na obr 4.7 jsou znázorněny průběhy signálů modulu `bin9_bcd.vhd`. Tento modul je devítibitový paralelní převodník binárního kódu do kódu BCD.



Obr. 4.7 Simulační průběhy modulu BIN9_BCD

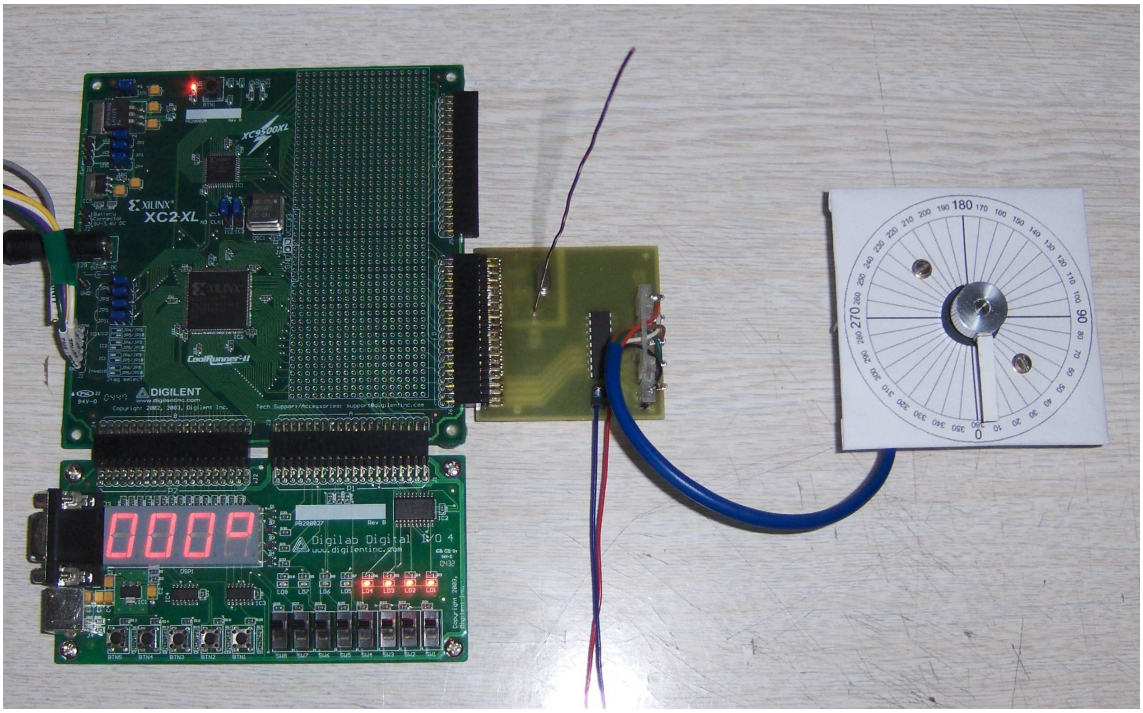
4.6 Praktická realizace

Na obr 4.8 je praktická realizace vysílací části. Vysílací část je připojena pomocí rozhranní RS232 k PC. Rozhranní USB slouží pouze k získání napájecího napětí.



Obr. 4.8 Praktická realizace vysílací části

Na obr. 4.9 je praktická realizace přijímací části. K vývojové desce je připojeno navržené přijímací zařízení. K zařízení je připojen krokový motor se stupnicí ukazující natočení hřídele. Přijatá hodnota je zobrazena na displeji rozšiřující desky.



Obr. 4.9 Praktická realizace přijímací části

5. ZÁVĚR

Byl navržen obvod pro řízení otáčení unipolárního krokového motoru. Řízení lze provádět polovičním nebo plným krokem oběma směry. Pokud nebude krokový motor přetěžován, není nutné doplňovat zpětnou vazbu o úhlu natočení. Jinak se zpětná vazba realizuje optickou závorou. Program je na tuto variantu připraven. Řídící obvod byl doplněn o radiový přijímací obvod přijímající data z řídícího počítače. Kvůli kontrole funkce byl doplněn displej zobrazující přijatou polohu. Jednosměrná komunikace je zajištěna pomocí RF obvodů. RF obvody jsou citlivé na okolní rušení. Tento problém byl vyřešen kódováním datového rámce a několikanásobným vysláním dat. Celkové zapojení anténního rotátoru bylo vyzkoušeno a odladěno. Byly navrženy desky plošných spojů pro vysílací a přijímací část. Program pro ovládání z řídícího počítače lze použít libovolný kompatibilní s rozhranním RS232. Pro ukázkové a testovací účely byl navržen program vlastní.

LITERATURA

[1] OBDRŽÁLEK J. DUB P. a jiní. Vysokoškolská učebnice obecné fyziky HRW. Brno: VUTIUM, 2000. 562 stran. ISBN 80-214-1869-9.

[2] TKOTZ, K. , HANDLÍŘ, J. a jiní. Příručka pro elektrotechniky. Praha: Europa-Sobotáles cz, 2001. 562 stran. ISBN 80-86706-00-1.

[3] ŘEZÁČ, K. Krokové motory [online]. 2002 – [cit. 20.11.2008]. Dostupné na WWW: <http://robotika.cz/articles/steppers/cs>.

[4] SINGULE, V. Vlastnosti a použití mikromotorů [online] – [cit. 20.11.2008]. Dostupné na WWW: <http://www.odbornecasopisy.cz/index.php?id_document=36910>.

[5] POUPA, M. Přednášky z předmětu Programovatelné logické obvody [online] –[cit.5.12.2008]. Dostupné na WWW:<www.micro.feld.cvut.cz/home/hazdra/x/prednasky/Prednaska7.pdf>.

[6] MUSIL, V. a jiní. Návrh digitálních integrovaných obvodů VLSI a jazyk VHDL [Skriptum VUT Brno.] PC DIR, Brno 2002.

[7] HAZRDA, P. Přednáška Jazyk VHDL [online] – [cit. 5.12.2008]. Dostupné na WWW: <http://measure.feld.cvut.cz/groups/edu/x38aph/pdf/VHDL.pdf>.

[8] CAVALLI, C. Serial Communication with VB.Net [online] – [cit.16.5.2009]. Dostupné na WWW: <http://www.codeworks.it/net/VBNetRs232.htm>.

[9] Micrel Inc., Katalogový list obvodu MICRF102 [online] – [cit.16.5.2009]. Dostupné na WWW: < http://www.datasheetcatalog.org/datasheet/Micrel/mXsvwwu.pdf>.

[10] Micrel Inc., Katalogový list obvodu MICRF007 [online] – [cit.16.5.2009]. Dostupné na WWW: <http://www.datasheetcatalog.org/datasheet/Micrel/mXsvwww.pdf>.

[11] LOOMIS, J. Binary to BCD Converter [online] – [cit.16.5.2009]. Dostupné na WWW: <http://www.engr.udayton.edu/faculty/jloomis/ece314 /notes/devices/binary_to_BCD/bin_to_BCD.html>.

[12] ČTÚ. Využívání radiových kmitočtů vymezené všeobecným oprávněním [online] – [cit.20.5.2009]. Dostupné na WWW:<http://www.ctu.cz/ctu-informuje/jak-postupovat/ radiove-kmitocty/vyuzivani-vymezenych-radiovych-kmitoctu.html>.

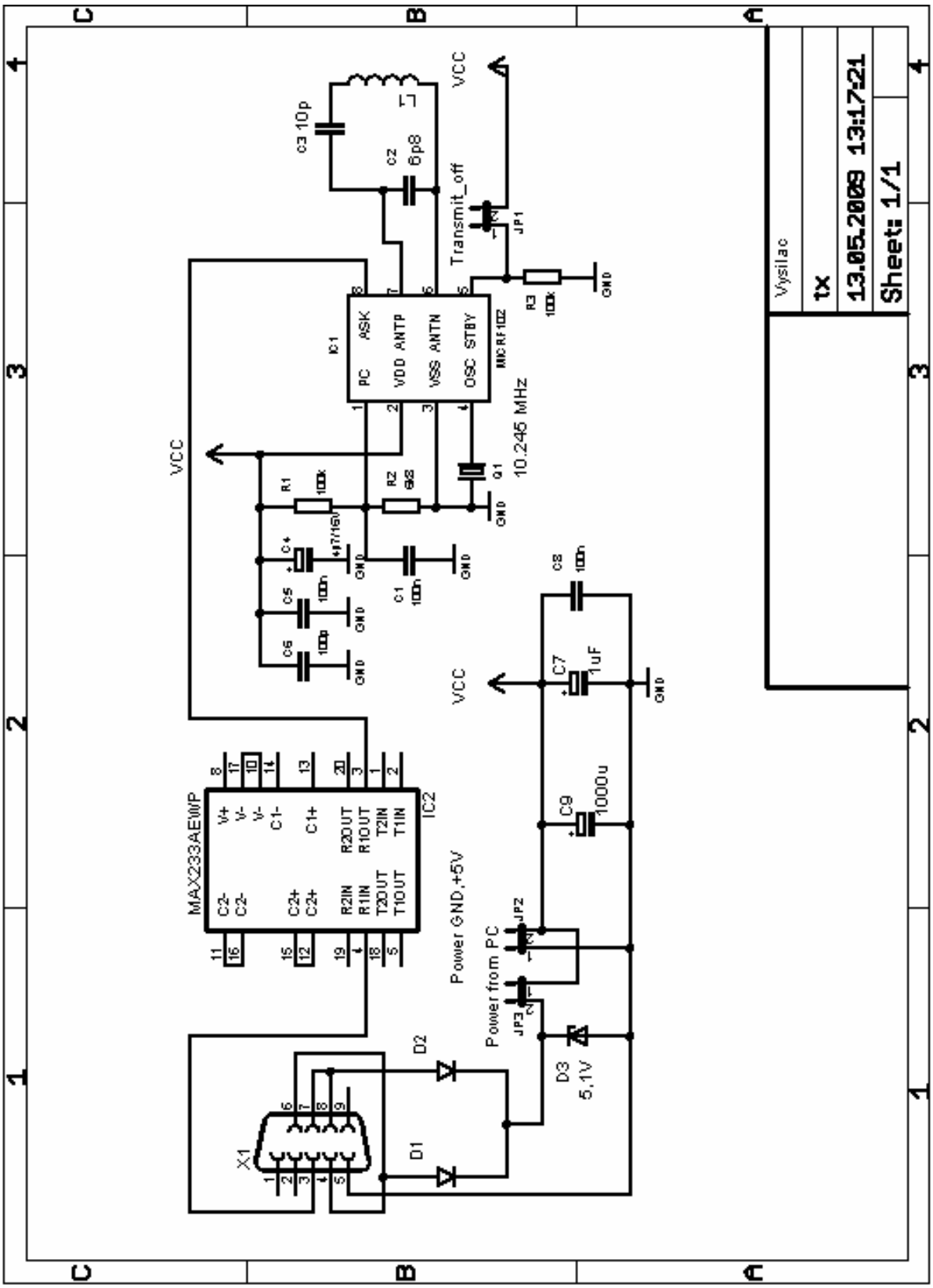
SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

PLD	Programmable Logic Device (Programovatelné Logické Zařízení)
PROM	Programmable Read Only Memory (Programovatelná Paměť Jen pro Čtení)
PLA	Programmable Logic Array (Programovatelné Logické Pole)
PAL	Program Array Logic (Programovatelné Logické Pole)
GAL	Generic Array Logic (Obecné Logické Pole)
SPLD	Simple Programmable Logic Device (Jednoduché Programovatelné Logické Obvody)
CPLD	Complex Programmable Logic Device (Komplexní Programovatelné Logické Zařízení)
FPGA	Field Programmable Gate Array (Programovatelná hradlová pole)
RAM	Random Access Memory
PLL	Phase-locked loop (Smyčka Fázového Závěsu)
ABEL	Assumption Based Evidential Language
VHDL	akronym VHSIC HDL
VHSIC HDL	Very High Speed Integrated Circuits Hardware description language. (Velmi rychlé integrované obvody Jazyk popisující hardware)
RF	Radio Frequency
LUT	LookUp Table
JTAG	Joint Test Action Group (Rozhraní pro programování a testování obvodů)
ISP	In System Programing (Programování v cílovém systému)
IEEE	The Institute of Electrical and Electronics Engineers
MC	Macrocell (Makrobuňka)
LED	Light Emitted Diode (Světlo emitující dioda)
PS/2	Personal System /*2* (Konektor mini-din pro připojení myši nebo klávesnice)
VGA	Video Graphic Array (Konektor pro připojení monitoru)
Tx	Transceiver (Vysílač)
Rx	Receiver (Přijímač)
UHF	Ultra High Frequency (Ultra krátké vlny 300 – 3000 MHz)
TTL	Tranzistor Tranzistor Logic (Tranzistorová logika)
RS232	Recommended Standard 232 (Standard pro sériovou komunikaci)
PC	Personal computer (Osobní počítač)
UART	Universal Asynchronous Receiver/Transmitter (Univerzální Asynchronní Vysílač / Přijímač)

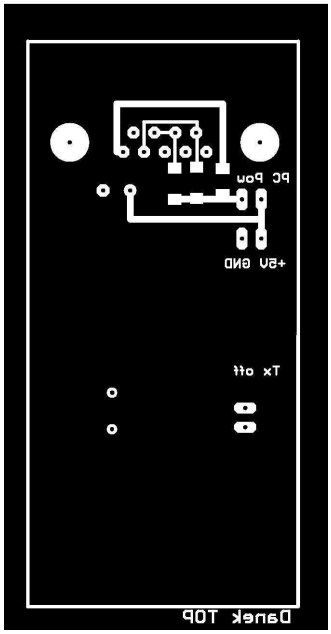
SEZNAM PŘÍLOH

A NÁVRH ZAŘÍZENÍ VYSÍLACÍ ČÁSTI

A.1 Obvodové zapojení vysílací části

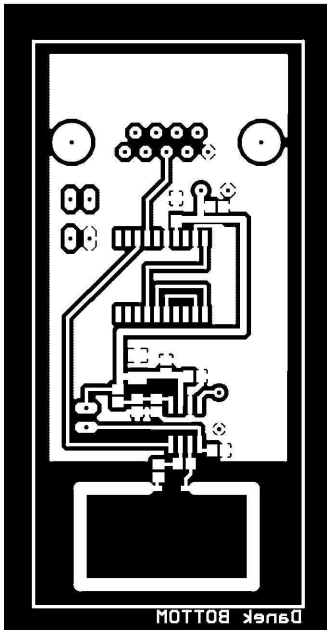


A.2 Deska plošného spoje – strana součástek



Rozměr desky 35 x 75 [mm], měřítko M1:1

A.3 Deska plošného spoje – strana spojů



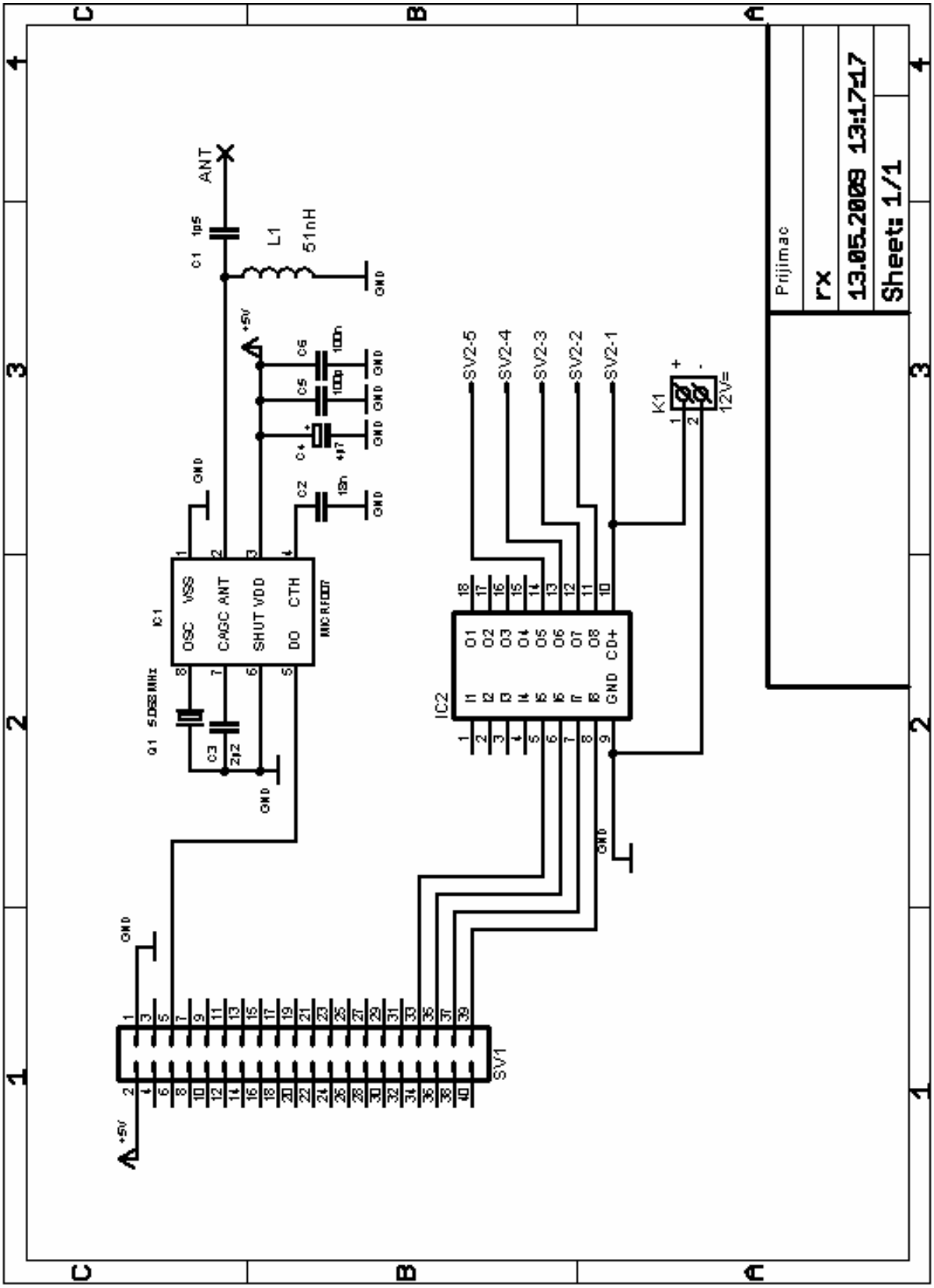
Rozměr desky 35 x 75 [mm], měřítko M1:1

A.4 Seznam součástek vysílací části

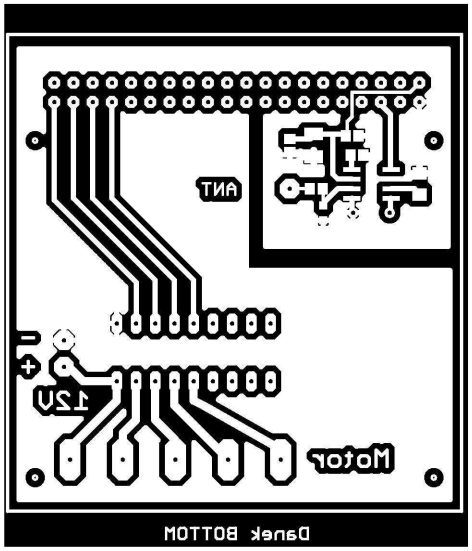
Označení	Hodnota	Pouzdro	Popis
C1	100n	C0805	Keramický kondenzátor
C2	6p8	C0805	Keramický kondenzátor
C3	10p	C0805	Keramický kondenzátor
C4	4µ7/16V	SMC_B	Keramický kondenzátor
C5	100n	C0805	Keramický kondenzátor
C6	100p	C0805	Keramický kondenzátor
C7	1uF	SMC_A	Keramický kondenzátor
C8	100n	C0805	Keramický kondenzátor
C9	1000u	E3,5-8	Keramický kondenzátor
D1	1N4148	SMD_SOD80	Schottkyho dioda
D2	1N4148	SMD_SOD80	Schottkyho dioda
D3	5,1V	SMD_SOD80	Zenerova dioda
IC1	MICRF102	SOIC8	Vysílač
IC2	MAX233AEWP	SO20L	Převodník RS232-CMOS
JP1	Transmit_off	1X02	Propojka
JP2	Power GND,+5V	1X02	Propojka
JP3	Power from PC	1X02	Propojka
L1	LANT	PCB	Smyčková anténa
Q1	10.245 MHz	HC49U-H	Krystalový oscilátor
R1	100k	R0805	Rezistor
R3	6k8	R0805	Rezistor
R3	100k	R0805	Rezistor
X1	F09HP	F09HP	RS 232 konektor

B NÁVRH ZAŘÍZENÍ PŘIJÍMACÍ ČÁSTI

B.1 Obvodové zapojení přijímací části



B.2 Deska plošného spoje – strana spojů



Rozměr desky 60 x 61 [mm], měřítko M1:1

B.3 Seznam součástek vysílací části

Označení	Hodnota	Pouzdro	Popis
ANT	Délka 23,1 cm		Drátová anténa
C1	1p5	C0805	Keramický kondenzátor
C2	18n	C0805	Keramický kondenzátor
C3	2µ2	C1206	Keramický kondenzátor
C4	4µ7/16V	SMC_B	Keramický kondenzátor
C5	100p	C0805	Keramický kondenzátor
C6	100n	C0805	Keramický kondenzátor
IC1	MICRF007	SOIC8	Přijímač
IC2	ULN2803A	DIL18	Spínací tranzistor 8x
L1	51nH	L2012C	Cívka
Q1	5.068 MHz	HC49/S	Krystalový oscilátor
K1	100k	R0805	Odpor
SV1		MA20-2W	Vidlice 40 pin
SV2		con-WK180	Vidlice 5 pin

C ČÁST ZDROJOVÉHO KÓDU RS232

```
Private Sub Button1_Click_1(ByVal sender As System.Object, ByVal e
                            As System.EventArgs) Handles btnTx.Click
        Dim i, index As Integer
        Dim a As Byte
        index = CInt(Val(repeat.Text)) * 4 - 1
        ReDim Out(index)
        '// Clear Tx/Rx Buffers
        moRS232.PurgeBuffer(Rs232.PurgeBuffers.TxClear Or
                            Rs232.PurgeBuffers.RXClear)
        a = CByte(Val(txtTx.Text))
        'vytvori jeden povel
        out(0) = 171
        out(1) = a
        Out(2) = CByte(255 - a)
        Out(3) = 171
        'posle nekolikrat za sebou
        While i < index + 1
            Out(i) = Out(0)
            Out(i + 1) = Out(1)
            Out(i + 2) = Out(2)
            Out(i + 3) = Out(3)
            i = i + 4
        End While
        moRS232.Write(out)
    End Sub

Public Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles Timer1.Tick, chkAutorx.CheckedChanged
        Dim i, index As Integer
        index = CInt(Val(repeat.Text)) * 4 - 1
        ReDim Out(index)
        '// Clear Tx/Rx Buffers
        moRS232.PurgeBuffer(Rs232.PurgeBuffers.TxClear Or
                                    Rs232.PurgeBuffers.RXClear)
        If chkrandom.Checked Then
            a = CByte((199 + 1) * Rnd())
            runstep.Text = Str(a)
        Else
            a = a + CByte(Val(runstep.Text))
            If a > 198 Then
                a = 0
            End If
        End If
        'vytvori jeden povel
        out(0) = 171
        out(1) = CByte(a)
        Out(2) = CByte(255 - a)
        Out(3) = 171
        'posle nekolikrat za sebou
        While i < index + 1
            Out(i) = Out(0)
            Out(i + 1) = Out(1)
            Out(i + 2) = Out(2)
            Out(i + 3) = Out(3)
            i = i + 4
        End While
        moRS232.Write(Out)
    End Sub
```

D POPIS HARDWAROVÉ KONFIGURACE CPLD

D.1 Hlavní propojení (Main)

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use ieee.numeric_std.all;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 ---- Uncomment the following library declaration if instantiating
7 ---- any Xilinx primitives in this code.
8 --library UNISIM;
9 --use UNISIM.VComponents.all;
10
11 entity main is
12 Generic ( Clk_divMax : integer := 13);
13
14 Port ( output_mot : out STD_LOGIC_VECTOR (3 downto 0):="0000";
15        clk,sensor_en : in std_logic;
16        rx_out : out std_logic_vector(7 downto 0);
17        rx_data : in STD_LOGIC;
18        digit4: out std_logic_vector(3 downto 0);
19        seg4: out std_logic_vector(7 downto 0)
20 );
21 end main;
22 architecture Behavioral of main is
23
24 component rf
25 Port ( rx_out : out std_logic_vector(7 downto 0);
26        data_ready : out STD_LOGIC;
27        rx_data,rx_16clk : in STD_LOGIC);
28 end component ;
29
30 component rotator
31 Port ( output_mot : out STD_LOGIC_VECTOR (3 downto 0):="0000";
32        clk : in STD_LOGIC;
33        position : in STD_LOGIC_VECTOR (7 downto 0); -- pozice 0-199
34        position_en : in STD_LOGIC; -- spousti toceni na pozici
35        sensor_en : in STD_LOGIC); -- imp ze senzoru polohy
36 end component ;
37
38 component DspDr
39 Port(  clk: in std_logic;
40        bcdint: in std_logic_vector(11 downto 0);
41        dp3,dp2,dp1,dp0: in std_logic;
42        digit: out std_logic_vector(3 downto 0);
43        seg: out std_logic_vector(7 downto 0));
44 end component ;
45
46 component Bin9_BCD
47 PORT(Bin: IN std_logic_vector(8 DOWNTO 0);
48      BCD: OUT std_logic_vector(11 DOWNTO 0));
49 end component;
50
51 signal rx_16clk,rot_clk,data_ready,clk32k : STD_LOGIC ;
52 signal clk_div :STD_LOGIC_VECTOR (Clk_divMax downto 0):= (OTHERS =>'0');
53 signal position :STD_LOGIC_VECTOR (7 downto 0);
54 signal bcdint: std_logic_vector(11 downto 0) := x"000";
55 signal cnt_32k: std_logic_vector(10 downto 0);
56 signal bin_deg: std_logic_vector(8 downto 0);
```

```
57
58 begin
59 u1 : rf PORT MAP (
60                             rx_out => position,
61                             data_ready => data_ready,
62                             rx_data => rx_data,
63                             rx_16clk => rx_16clk
64                     );
65
66 u2 : rotator PORT MAP (
67                             output_mot => output_mot,
68                             clk => rot_clk,
69                             position => position,
70                             position_en => data_ready,
71                             sensor_en => sensor_en
72                     );
73
74 u3 : DspDr PORT MAP (
75                             clk=>clk,
76                             bcdint=>bcdint,
77                             dp3=>'1',
78                             dp2=>'1',
79                             dp1=>'1',
80                             dp0=>'1',
81                             digit=>digit4,
82                             seg=>seg4
83                         );
84
85 u4 : Bin9_BCD PORT MAP (
86                             Bin=>bin_deg,
87                             BCD=>bcdint
88                           );
89
90 process (clk)
91 begin
92 if clk='1' AND clk'event then
93          clk_div <= clk_div + 1;
94          cnt_32k <= cnt_32k + 1;
95      -- pro 32 khz / 16 / 2 = 2khz 1D 3A
96    if cnt_32k <= X"1D" then
97          clk32k <= '1';
98    else
99          if cnt_32k =X"3A" then
100              cnt_32k <= (others=> '0');
101              clk32k <= '0';
102         end if;
103   end if;
104 end if;
105 end process;
106
107 Bin_Deg <= std_logic_vector( to_unsigned( to_integer(unsigned('0'&
                                                  position) * 231/128),));
108 -- Jeden krok ma 1,804 stupne = 231/128
109 -- Převod na stupne v rozsahu 0-360,prijatelná presnost +-1
110
111 rx_out <= position;
112 -- Kmitočty hodinových signálů
113 rx_16clk<=clk32k;
114 rot_clk <= clk_div(clk_divMax); --1,8432M / 2na(13+1)=112,5 Hz
115
116 end Behavioral;
```

D.2 Radiový přijímač (u1 – rf)

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 ---- Uncomment the following library declaration if instantiating
7 ---- any Xilinx primitives in this code.
8 --library UNISIM;
9 --use UNISIM.VComponents.all;
10
11 -- Format přenosu: [AB][start bit][stop bit][slovo][start bit][stop bit]
                      [slovo negovane][stop bit][start bit][AB][stop bit]
12 entity rf is
13 Generic (
14    prfx : std_logic_vector(7 downto 0) := x"AB" -- AB kontrolni slovo
15 );
16
17 Port ( rx_out : out std_logic_vector(7 downto 0);
18        data_ready : out STD_LOGIC;
19        rx_data,rx_16clk : in STD_LOGIC);
20
21 end rf;
22
23 architecture Behavioral of rf is
24 signal regrx : std_logic_vector(39 downto 0) := (others => '0');
25 signal rx_clk : std_logic;
26
27 signal rxd1 : std_logic ;
28 signal rxd2 : std_logic ;
29 signal clkdiv : std_logic_vector (3 downto 0) := "0111" ;
30 signal edge : std_logic := '0' ;
31 begin
32 data_ready <= rx_clk ; --nutne, nechyta se na jednu hranu
33 rx_clk <= clkdiv(3) ;
34
35 process (rx_16clk,rxd1,rxd2)
36 begin
37 if rx_16clk'event and rx_16clk = '1' then
38    if rxd1 /= rxd2 then
                -- detekce hrany, vzorky odebirany v case 1/rx_16clk
39          edge <= '1';
40    else
41          edge <= '0';
42    end if ;
43 end if ;
44 end process ;
45
46 process (rx_16clk,edge)
47 begin
48 if rx_16clk'event and rx_16clk = '1' then
49    rxd1 <= rx_data ;
50    rxd2 <= rxd1 ;
51    clkdiv <= clkdiv + "0001" ;
52    if edge = '1' then
53          clkdiv <= "0011" ;
54    end if ;
55 end if ;
56 end process ;
```

```
57
58 ---- prijimac
59 process (rx_clk,rx_data,regrx) begin library IEEE;
60 if rx_clk'event and rx_clk='1' then
61    regrx <= (rx_data & regrx(39 downto 1));
      -- zkontroluje prefix a stop,start bity
62    if regrx(8 downto 1) = prfx and regrx(10 downto 9)="01" and
63     regrx(20 downto 19)="01" and regrx(30 downto 29)="01" and
       regrx(0)='0' and regrx(39) = '1' and regrx(38 downto 31) = prfx then
64
65 -- regrx(18 downto 11); --prima data
66 -- regrx(28 downto 21); --negovana data
67          if regrx(18 downto 11) = not regrx(28 downto 21) then
68                rx_out <= regrx(18 downto 11);
69                --na vystup zapise prima data
70          end if;
71    end if;
72 end if;
73 end process;
74
75 end Behavioral;
```

D.3 Otáčení krokovým motorem (u2 - rotator)

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity rotator is
7 Generic (
8           Sensor_position : integer := 0;
9 --Poloha senzoru v krocich vzhledem k pocatku, jeden krok 1,8 stupnu
10          StepsMax : integer := 199
11 --Maximalni pocet kroku
12 -- kroky 0 az 199, 360/1.8=200 kroku 8bitu, plny krok
13 -- (StepsMax+1)/2 = 100 odpovida 180 stupnum pro 0-199 kroku
14
15          );
16 Port (    output_mot : out STD_LOGIC_VECTOR (3 downto 0):="0000";
                                                        --neaktivní uroven
17          clk : in STD_LOGIC;
18          position : in STD_LOGIC_VECTOR (7 downto 0); -- pozice 0-199
19          position_en : in STD_LOGIC;       -- spousti toceni na pozici
20          sensor_en : in STD_LOGIC          -- imp ze senzoru polohy
21
22          );
23
24 end rotator;
25
26 architecture Behavioral of rotator is
27 -- Signaly stavoveho automatu
28 type ST_type is (s0,s1,s2,s3,s4,s5,s6,s7,s8);
29 signal now_ST : ST_type := s0;                --pocatecni stav s0
30 signal next_ST : ST_type := s1;
31 signal full_half : STD_LOGIC :='0';           -- 1 full step 0 half step
32 signal forward_backward : STD_LOGIC :='1';    -- 1 forward 0 backward
33 signal run : STD_LOGIC:='0';                  -- 1 run 0 stop
34 signal to_out :STD_LOGIC_VECTOR (3 downto 0):= (OTHERS => '0');
35 -- Signaly citace
36 signal cnt_catch :integer range 0 to StepsMax := 0; --poloha ze vstupu
37 signal cnt_reg :integer range 0 to StepsMax := 0;   --poloha je 0 až 199
38 signal sensor_set : STD_LOGIC := '1';
39
40 begin
41
42 -- Rozhodnuti o nataceni
43 -- Urceni smeru toceni a cilove pozice
44 process (position_en,sensor_en,cnt_catch,position,cnt_reg)
45 begin
46 if position_en='0' AND position_en'event then
47    if conv_integer(position) <= StepsMax then
                                                -- osetreni vstupni hodnoty
48          cnt_catch <= conv_integer(position); -- ulozi cilovou pozici
49                                                -- urci smer toceni
50
51    if abs(conv_integer(position) - cnt_reg) < ((StepsMax +1)/2) then
52          if cnt_reg < conv_integer(position) then --uhel 0 az 180
53                forward_backward <= '1'; --tocim dopredu
54          else
55                forward_backward <= '0'; --tocim dozadu
56          end if;
```

```
57    else
58          if conv_integer(position) < cnt_reg then --uhel 180 az 360
59                forward_backward <= '1' ; --tocim dopredu
60          else
61                forward_backward <= '0' ; --tocim dozadu
62          end if;
63    end if;
64    end if;
65 end if;
66 if sensor_en = '1' then
67    if abs(cnt_catch - sensor_position) < ((StepsMax +1)/2) then
68          if sensor_position < cnt_catch then --uhel 0 az 180
69                forward_backward <= '1'; --tocim dopredu
70          else
71                forward_backward <= '0'; --tocim dozadu
72          end if;
73    else
74          if cnt_catch < sensor_position then --uhel 180 az 360
75                forward_backward <= '1' ; --tocim dopredu
76          else
77                forward_backward <= '0' ; --tocim dozadu
78          end if;
79    end if;
80 end if;
81 end process;
82
83 --Synchronni cast,nastaveni dalsiho stavu,aktualni pozice
84 process (clk,run,position_en,next_ST,sensor_en,sensor_set)
85 begin
86 if sensor_en ='1' and sensor_set = '1' then -- když je signal ze senzoru
87    cnt_reg <= sensor_position;               -- upravi polohu
88    sensor_set <= '0';
89 else
90    if ( clk= '1' AND clk'event ) then
91
92      if (position_en ='1' OR run ='1') then
                                                -- pokud je run a position_en
93        run <='1';                            -- jede i dale
94
95        now_ST <= next_ST;                    -- posun stav
96
97        if (cnt_reg /= cnt_catch ) then      -- a pokud jsou pozice ruzne
98          if forward_backward = '1' then
                                              -- aktualizuj pozici v registru
99             cnt_reg <= ( cnt_reg + 1 ) ;
100            if cnt_reg = StepsMax then -- misto MOD StepsMax + 1
101               cnt_reg <= 0;
102            end if;
103         else
104            cnt_reg <= ( cnt_reg - 1 ) ;
105            if cnt_reg = 0 then
106               cnt_reg <= StepsMax;
107            end if;
108         end if;
109       else           -- kdyz jsou stejne pozice byl tohle poslední krok
110         sensor_set <= '1';
111         run <='0';
112         now_ST <= now_ST;                        -- neztrati jeden stav
113       end if;
114     end if;
115   end if;
```

```
116 end if;
117 end process;
118
119 -- Přirazeni vystupu
120 process (to_out,run)
121 begin
122 if run = '1' then
123   output_mot <= to_out;
124 else
125   output_mot <= "0000";
126 end if;
127
128 end process;
129
130 -- Kombinacni cast,nastaveni dalsiho stavu,prirazeni vystupu
131 --Unipolární řízení s polovičním krokem
132 --Cívka 1 - - 0 0 0 0 0 -
133 --Cívka 2 0 - - - 0 0 0 0
134 --Cívka 3 0 0 0 - - - 0 0
135 --Cívka 4 0 0 0 0 0 - - -
136 ----------------------------------------
137 -- full step forward s1 s3 s5 s7
138 -- full step backward s7 s5 s3 s1
139 -- half step backward s8 s7 s6 s5 s4 s3 s2 s2
140 -- half step forward s1 s2 s3 s4 s5 s6 s7 s8
141 process (now_ST,full_half,forward_backward,run)
142   variable aditional: STD_LOGIC_VECTOR (1 downto 0);
143 begin
144 aditional := full_half & forward_backward;
145 next_ST <= s1; --odstraneni latchu
146 case now_ST IS
147   when s0 => to_out <= "0000"; --odpojeni vystupu
148   when s1 => to_out <= "1000";
149      case aditional is
150         when "00" => next_ST <= s8; --half step backward
151         when "10" => next_ST <= s7; --full step backward
152         when "01" => next_ST <= s2; --half step forward
153         when "11" => next_ST <= s3; --full step forward
154         when others => next_ST <= s0;
155      end case ;
156   when s2 => to_out <= "1100";
157      case aditional is
158         when "00" => next_ST <= s1; --half step backward
159         when "10" => next_ST <= s1; --full step backward
160         when "01" => next_ST <= s3; --half step forward
161         when "11" => next_ST <= s3; --full step forward
162         when others => next_ST <= s0;
163      end case ;
164   when s3 => to_out <= "0100";
165      case aditional is
166         when "00" => next_ST <= s2; --half step backward
167         when "10" => next_ST <= s1; --full step backward
168         when "01" => next_ST <= s4; --half step forward
169         when "11" => next_ST <= s5; --full step forward
170         when others => next_ST <= s0;
171      end case ;
172   when s4 => to_out <= "0110";
173      case aditional is
174         when "00" => next_ST <= s3; --half step backward
175         when "10" => next_ST <= s3; --full step backward
176         when "01" => next_ST <= s5; --half step forward
```

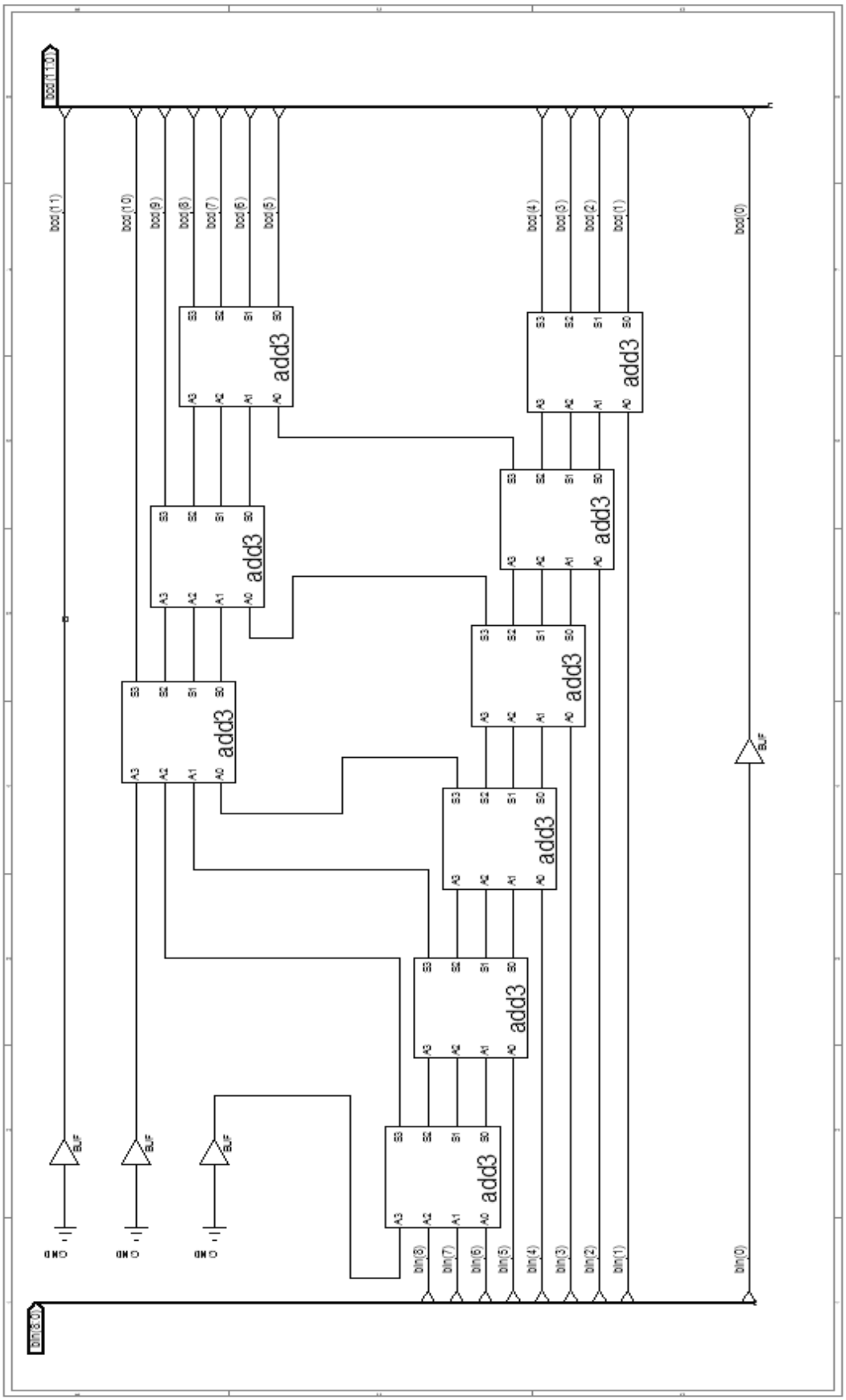
```
177         when "11" => next_ST <= s5; --full step forward
178         when others => next_ST <= s0;
179      end case ;
180   when s5 => to_out <= "0010";
181      case aditional is
182         when "00" => next_ST <= s4; --half step backward
183         when "10" => next_ST <= s3; --full step backward
184         when "01" => next_ST <= s6; --half step forward
185         when "11" => next_ST <= s7; --full step forward
186         when others => next_ST <= s0;
187      end case ;
188   when s6 => to_out <= "0011";
189      case aditional is
190         when "00" => next_ST <= s5; --half step backward
191         when "10" => next_ST <= s5; --full step backward
192         when "01" => next_ST <= s7; --half step forward
193         when "11" => next_ST <= s7; --full step forward
194         when others => next_ST <= s0;
195      end case ;
196   when s7 => to_out <= "0001";
197      case aditional is
198         when "00" => next_ST <= s6; --half step backward
199         when "10" => next_ST <= s5; --full step backward
200         when "01" => next_ST <= s8; --half step forward
201         when "11" => next_ST <= s1; --full step forward
202         when others => next_ST <= s0;
203      end case ;
204   when s8 => to_out <= "1001";
205      case aditional is
206         when "00" => next_ST <= s7; --half step backward
207         when "10" => next_ST <= s7; --full step backward
208         when "01" => next_ST <= s1; --half step forward
209         when "11" => next_ST <= s1; --full step forward
210         when others => next_ST <= s0;
211      end case ;
212 end case;
213 end process;
214
215 end Behavioral;
```

D.4 Obsluha displeje (u3 – DspDr)

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity DspDr is
7 port(clk: in std_logic;
8      bcdint: in std_logic_vector(11 downto 0);
9      dp3,dp2,dp1,dp0: in std_logic;
10     digit: out std_logic_vector(3 downto 0);
11     seg: out std_logic_vector(7 downto 0));
12 end DspDr;
13
14 architecture Behavioral of DspDr is
15 signal cd: std_logic_vector(1 downto 0);
16 signal curr: std_logic_vector(3 downto 0);
17 signal dp: std_logic;
18 SIGNAL mhertz_count: std_logic;
19 SIGNAL khertz_count: std_logic_vector(9 downto 0);
20 signal mhertz_en,khertz_en: std_logic;
21 begin
22
23 -- generates a 900 kHz signal from a 1.8 Mhz signal - mhertz_en
24 process (clk) begin -- description follows timing module for XC3S
25 if clk'event and clk = '1' then
26    mhertz_count <= NOT mhertz_count;
27    if mhertz_count = '1' then
28          mhertz_en <= '1' ;
29    else
30          mhertz_en <= '0' ;
31    end if ;
32 end if ;
33 end process ;
34
35 -- generates a 1 kHz signal from a 1Mhz signal - khertz_en
36 process (clk) begin
37 if clk'event and clk = '1' then
38    if mhertz_en = '1' then
39          khertz_count <= khertz_count + 1 ;
40          if khertz_count = "1111101000" then
41                khertz_en <= '1' ;
42                khertz_count <= (others => '0') ;
43          else
44                khertz_en <= '0' ;
45          end if ;
46    else
47          khertz_en <= '0' ;
48    end if ;
49 end if ;
50 end process ;
51
52 process (clk) begin
53 if clk'event and clk = '1' then
54   if khertz_en = '1' then
55          cd <= cd + 1;
56   end if;
57   case cd(1 downto 0) is
-- curr je soucasne zobrazena cifra, digit je jeji pozice na displeji
```

```
58
59    when "00" => curr <="1011"; digit <= "1110"; dp <= dp0;
60    when "01" => curr <= bcdint(3 downto 0); digit <= "1101"; dp <= dp1;
61    when "10" => curr <= bcdint(7 downto 4); digit <= "1011"; dp <= dp2;
62    when others => curr <= bcdint(11 downto 8);digit <= "0111";dp <= dp3;
63   end case ;
64   case curr is
65         when "0000" => seg <= "0000001" & dp; -- 0
66         when "0001" => seg <= "1001111" & dp; -- 1
67         when "0010" => seg <= "0010010" & dp; -- 2
68         when "0011" => seg <= "0000110" & dp; -- 3
69         when "0100" => seg <= "1001100" & dp; -- 4
70         when "0101" => seg <= "0100100" & dp; -- 5
71         when "0110" => seg <= "0100000" & dp; -- 6
72         when "0111" => seg <= "0001111" & dp; -- 7
73         when "1000" => seg <= "0000000" & dp; -- 8
74         when "1001" => seg <= "0000100" & dp; -- 9
75 --       when "1010" => seg <= "0001000" & dp; -- A
76 --       when "1011" => seg <= "1100000" & dp; -- b
77 --       when "1100" => seg <= "0110001" & dp; -- C
78 --       when "1101" => seg <= "1000010" & dp; -- d
79 --       when "1110" => seg <= "0110000" & dp; -- E
80 --       when others => seg <= "0111000" & dp; -- F
81         when others => seg <= "0011100" & dp; -- °
82   end case;
83 end if;
84 end process;
85 end Behavioral;
```

D.5 Převodník kódu bin do bcd (u4 - Bin9_Bcd)



D.6 Převodník kódu bin do bcd (add3)

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity add3 is
7 Port ( A0,A1,A2,A3 : in STD_LOGIC;
8        S0,S1,S2,S3 : out STD_LOGIC);
9 end add3;
10
11 architecture Behavioral of add3 is
12 begin
13
14 process(A0,A1,A2,A3)
15    variable A : std_logic_vector(3 downto 0);
16 begin
17 A := A3 & A2 & A1 & A0;
18 case A is
19    when "0000" => S3 <= '0';
20          S2 <= '0';
21          S1 <= '0';
22          S0 <= '0';
23    when "0001" => S3 <= '0';
24          S2 <= '0';
25          S1 <= '0';
26          S0 <= '1';
27    when "0010" => S3 <= '0';
28          S2 <= '0';
29          S1 <= '1';
30          S0 <= '0';
31    when "0011" => S3 <= '0';
32          S2 <= '0';
33          S1 <= '1';
34          S0 <= '1';
35    when "0100" => S3 <= '0';
36          S2 <= '1';
37          S1 <= '0';
38          S0 <= '0';
39    when "0101" => S3 <= '1';
40          S2 <= '0';
41          S1 <= '0';
42          S0 <= '0';
43    when "0110" => S3 <= '1';
44          S2 <= '0';
45          S1 <= '0';
46          S0 <= '1';
47    when "0111" => S3 <= '1';
48          S2 <= '0';
49          S1 <= '1';
50          S0 <= '0';
51    when "1000" => S3 <= '1';
52          S2 <= '0';
53          S1 <= '1';
54          S0 <= '1';
55    when "1001" => S3 <= '1';
56          S2 <= '1';
57          S1 <= '0';
58          S0 <= '0';
```

```
59    when others => S3 <= '0';
60          S2 <= '0';
61          S1 <= '0';
62          S0 <= '0';
63 end case;
64 end process;
65 end Behavioral;
```